USB Based Controller Interface Board Design

For the Pendubot

Li Zhang

A Thesis

In

The Department

Of

Mechanical and Industrial Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

March 2004

Your file Votre référence
ISBN: 0-612-91155-1
Our file Notre référence
ISBN: 0-612-91155-1

Abstract

USB Based Controller Interface Board Design for the Pendubot

Li Zhang

The Pendulum Robot is a benchmark mechatronic device for research in nonlinear control and robotics and for use in control engineering education. It consists of two rigid links interconnected by revolute joints. The first joint is actuated by a DC-motor and the second joint is under actuated. Using the Pendubot, one can investigate system identification, linear control, nonlinear control, optimal control, learning control, robust and adaptive control, fuzzy logic control, intelligent control, hybrid and switching control, gain scheduling, and others.

The purpose of this thesis is to design and build a new Pendubot controller interface board. This board will serve as an interface between the Pendubot and the PC, managing all data acquisition and digit to analog conversion. The approach taken in this design is to use USB interface for the device to communicate with the PC. In the thesis, a detailed description of circuit design for the Pendubot controller interface board is presented. The board has been designed that minimizes the amount of PC resource used, simplifies hardware installation, and reduces the cost.

A prototype of the controller interface board is made. Control algorithms for balancing the link at top and mid position are implemented on this new Pendubot controller interface board. The experimental results demonstrate that this newly designed controller interface board can work well and have high commercial values.

Acknowledgements

The author is greatly indebted to her supervisor Dr. Chun-Yi Su for his valuable guidance and support throughout the research and preparation of this thesis.

Greatly thanks to my husband Mr. Feng Zhao who helped me to perform the experiments during the development of this thesis.

Finally, I would like to thank my parents for their love and support throughout my life.

Table of Contents

List of Figure

List of Table

Nomenclature

D(q):	Inertia Matrix
C(q, $\dot{q}$):	Coriolis and Centrifugal Matrix
G(q):	Gravitational Vector
F(q):	Friction Vector
τ:	Input torque
q_1:	Angle of link 1
$\dot{q}_1$:	Velocity of link 1
q_2:	Angle of link 2
$\dot{q}_2$:	Velocity of link 2
λ(z(t)):	Membership function
P_1:	Positive Definite Matrix
X:	A neighborhood of the origin of R^n
Z:	A neighborhood of the origin of R^v
W:	A neighborhood of the origin of R^s
V:	Lyapunov Function
σ:	The spectrum

Chapter 1

Introduction

1.1 Background

Control design and analysis for underactuated mechanical system is currently an active field of research. The importance of the underactuated system is due to their broad applications in robotics, aerospace vehicles, and marine vehicles. In addition, restriction of the control authority in underactuated system offers challenging control problems from theoretical view.

The Pendulum Robot, as an underactuated system, is a mechatronic device for use in control engineering education and for research in nonlinear control and robotics. It consists of two rigid links interconnected by revolute joints. The first joint is actuated by a DC-motor and the second joint is underactuated. Because of the nonlinear dynamic coupling connection between two links, many fundamental concepts in nonlinear dynamics and control theory may be studied on this device.

Using the Pendubot one can investigate system identification, linear control, nonlinear control, optimal control, learning control, robust and adaptive control, fuzzy logic control, intelligent control, hybrid and switching control, gain scheduling, and other control paradigms. One can program the Pendubot for swing-up control, balancing, regulation and tracking, identification, gain scheduling, disturbance rejection, and friction compensation.

A Pendubot Model P-2 system was bought from Mechatronic Systems Incorporated (MSI) for our experiment and research purpose. MSI provides all necessary parts to assemble the Pendubot. Customer needs a little hardware knowledge to set it up.

MSI also provides eight controllers software for testing the system. By using these controllers, we found most of Pendubot control operation, such as swinging up and balancing the Pendubot in the mid and top point, can easily be achieved on the Pendubot system. Because MSI has given all necessary mathematical model and system parameters to be used for the control experiment, user can implement their own controller algorithm on the Pendubot system easily.

When we use the Pendubot experimental system, we also encounter some drawbacks. First of all, almost all of our new computers could not be used to serve as a central controller in the Pendubot system. The reason is that the Pendubot system requires a computer with at least three ISA bus slots to install its interface cards, but starting in the early 90s, ISA bus began to be replaced by the PCI local bus architecture and the most computers made today do not include ISA bus any more. Second, due to all application software and hardware drivers can only run under old operating system such as DOS or Window 3.1 and Window 95, it makes very inconvenient for user who already is familiar with Window98, Window 2000, or Window XP.

In this project, a new controller interface board is proposed and designed to replace the old one used in Pendubot in order to overcome previous problem.

1.2 Objectives:

The purpose of this project is to design and build a Pendubot controller interface board that will enable to communicate with host PC and manage all data acquisition and transmission.

1.3 Thesis Layout

Chapter two gives a general description of Pendubot. It also gives a quick review of Pendubot control system about previous design. An overall control flow chart is described as well.

In chapter three, the functionality of each component in the control system is discussed. Design specification is outlined based on the requirement of Pendubot system.

In chapter four, a proposed controller interface board hardware design is introduced. The controller interface board consists of four parts, i.e. USB interface, microprocessor, D/A converter and encoder counter. Some possible design options for each main functional block are discussed in this chapter. Some component test and verification are also done to ensure the components could meet design requirement.

Chapter five explains the detail hardware and software design of the new controller interface board. Each individual component is analyzed with respect to its main features, functionality and chip implementation. A circuit level design is demonstrated in this chapter with detail schematic diagrams. Software design is also explained in pseudo code. Reference is made to source code files that can be found in appendix.

In chapter six, the experiment tests are conducted to verify that the new controller interface board could function properly, which include I^2C communication test and D/A converter calibration. The test methods are discussed in this chapter. All the software for the test purpose are provided and test result diagram are given.

Chapter seven discussed the Pendubot mathematical model and identification of the parameters. Two classical control algorithms used to swing up and balance the links

at unstable equilibrium point are given. The control algorithm is used to test the new controller interface board performance.

1.4 Contributions

A new Pendubot controller interface board has been proposed and designed. A detail circuit design of the board in according with user software and hardware firmware is provided in this thesis. A prototype of board is made and its performance was tested and verified. A classical Pendubot control algorithm to balance the link at top and mid position was running on this new controller interface board. The experimental results have been very good.

Chapter 2

Pendubot System

2.1. Introduction

The Pendubot, short for **PENDU**lum **ROB**ot, is a two links planar robot with an actuator at the shoulder but no actuator at the elbow. The actuated joint is driven by DC-motor. Thus the no actuator link is like as a simple pendulum whose motion is controlled by actuation of the first link. Because of the nonlinear dynamic coupling connection between two links, many fundamental concepts in nonlinear dynamics and control theory may be studied on this device.

2.2. Hardware Description

A Pendubot control experiment system designed by Mechatronic System Incorporated (MSI) is illustrated in Figure 2.1.

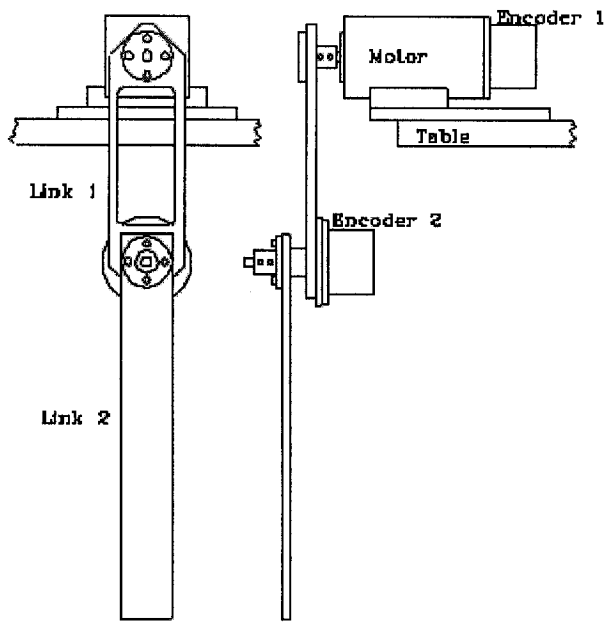

Figure2.1 Pendubot

The Pendubot consists of two rigid aluminum links machined from ¼ inch thick aluminum. The first link is 6 inch long and is directly coupling to shaft of a 90V permanent magnet DC motor at one end. The motor is mounted on the end of a table. At another end, the link has the bearing housing which allows for the motion of joint two. The shaft extends out of both directions of the housing; allowing the coupling to attach both two links and an optical encoder mounted on link one. The second link is 9 inch long and contains a coupling that attached to shaft of joint two. The design gives both links full 360 of rotational motion.

Two 1024 counts /rev resolution optical encoders of Dynamics Research Corporation provide position feedback, one attached at the elbow joint and one attached to the motor. A PWM servo amplifier, which is made by Advanced Motion Controls, is used to drive the motor.

The amplifier is used in torque model and adjusted for a gain of 1V=1.2 Amps. In the control algorithm, this amplifier can be considered as a gain.

2.3. Control System Description

A computer with a D/A card and an encoder interface card is served as the controller in Pendubot control system. The control system diagram is given in Figure 2.2.

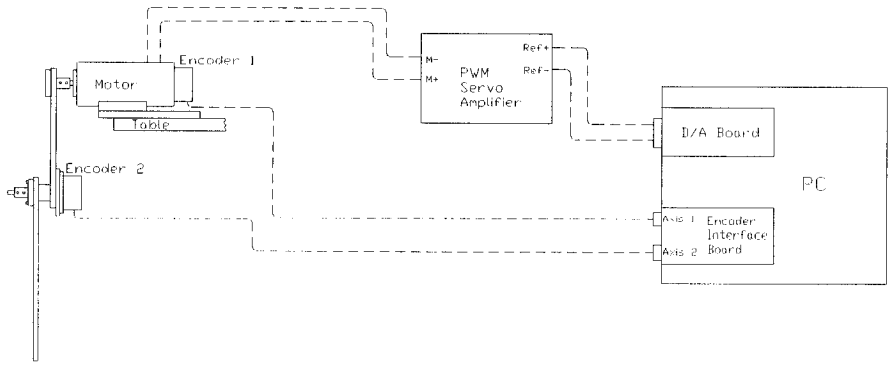

Figure 2.2 Pendubot control system

In this system, two 1024 counts/rev resolution optical encoders are used as the feedback mechanism for the joint angle. The output of these optical encoders is square wave signal. The encoder interface card is used to count the signals and then transmit the direction and position information of links to computer. The computer serves as a digital controller and performs control computation according to the control algorithms resided inside. The output of computer is Pendubot digital control signal. By using a D/A converter card, the digital signal is converted to standard −10V to +10V analog control signal. This is amplified by PWM servo amplifier and then used to drive motor.

Because DOS system does not produce a fast enough clock pulse, an extra timer card is also used to provide controller timing.

Chapter 3

New Controller interface board Design and Specification

3.1. Introduction

Although the MSI Pendubot experimental system can run very well and satisfy the most of experiment and research requirement, its disadvantage is also obvious. As mentioned before, there are three additional cards used in this control system. They all use ISA bus to interface with computer. Since Pentium II came into market, more and more new computers have not supported ISA interface bus. Therefore, it becomes impossible to use new and fast computer to control MSI Pendubot system. In this project, a new controller interface board is designed while its mechanical design remains same.

In the previous design, the main Pendubot controller is a PC computer with three additional ISA cards, i.e. CIO-DAC02 card, CIO-CTR05 card and a DRC encoder card. The CIO-DAC02 is a 12-bit digital to analog converter card; CIO-CTR05 is a real time counter card that serves as an accurate clock; DRC is used to read the square wave signals created by two optical encoders in the system as a 24-bit counter.

The CIO-DAC02 provides two channel D/A converters, which can be used to control voltage device in the range of 0--5V, 0--10V, -5V--+5 V and –10V--+10V voltages and 4--20mA electric current output. In additional, it provides +5V, -5V, +10V and -10V reference voltages. In MSI control system, one D/A converter channel with -10V to +10V voltage output is required only. No reference voltage output was used [14].

CIO-CTR05 time counter card has five 16-bit counters with on-board XTAL=1 MHz. It may be configured for event counting, pulse width measurement, frequency measurement, frequency division, generating complex duty cycles and much more. It

supports one-shot and continuous modes. In MSI design, only one 16-bit counter is used. The counter has ability up to 1ns resolution clock counter.

The DRC board has two individual Up/Down counters and allows two incremental shaft encoders to connect with it at same time. It is available to be extended to six input ports. The Up/Down counter is consisted by three 8-bit registers up to 24 bits.

The advantages of the design are:

1) All interface cards can get easily in market
2) No additional calibrations or tunings are required for the cards
3) Because the cards interface into computer directly with ISA bus, computer can access the cards easily by system interrupt. Therefore the data communication speed is fast

The disadvantages of the design are:

1) No ISA bus slots are provided in recent new model computer.
2) User needs a little computer hardware knowledge to install these three interface cards into PC.
3) Need install all drivers and make proper hardware and software setting for each card.
4) Most of features on these boards are not used.
5) Different features provided from three different cards will increase more defect possibilities.
6) The whole control system is expensive.

The new controller interface board will be different from previous design. It could be interfaced with computer easily without any restriction. The hardware design for

controller will be more simplifier than before. All hardware, such as D/A converter, encoder counter and clock counter, will be integrated into one controller interface board. This new controller interface board will perform either data acquisition or data transmission. A higher speed interface is required in order to build an efficiency communication between the PC and this new board.

3.2. Function Requirement

3.2.1. Encoder Counter

As described in the previous chapter, the Pendubot system uses two optical shaft encoders to read the position of the two links. Both of the encoders have 1024 counts/rev. A shaft encoder is an electromechanical transducer, which can convert a rotary position into an electronic signal. Each encoder has two output channels. Their output signals are shown in Figure 3.1.

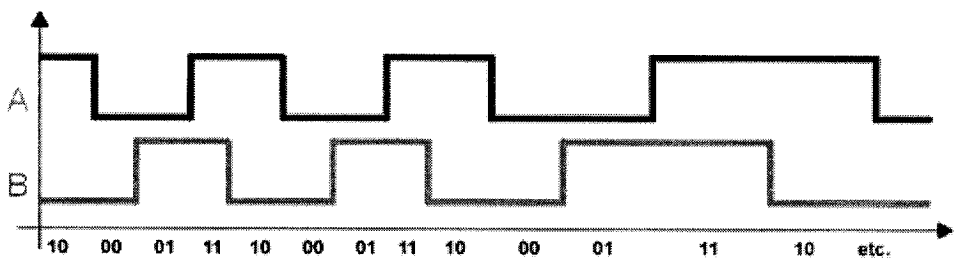

Figure 3.1 Encoder output signal

As shown in the Figure 3.1, signals A and B are square wave with 90° out of phase. If a counter can accept direction-sensing pulse signal, the resolution of the encoders can be increased four times to 4096 counts/rev. Therefore an encoder counter

reader in new design should have quadrature mode feature with over 12-bit counter (2^{12}=4096).

3.2.2. D/A Converter

In order to control the 90V DC motor, a D/A converter is required to convert a digital signal to an analog signal first. After that, the analog signal needs to be amplified by a servo amplifier and then used to drive motor. Because the accuracy of a controller relative to the resolution of its D/A converter, the resolution of D/A converter should be chosen as high as possible. Considering the performance and expense, 12-bit D/A converter is a good choice in new design. Because the motion of link can rotate in both directions, a D/A converter should works in bipolar operation mode with voltage output range from –10V to +10V.

3.2.3. Timer counter

The timer counter is a free-run real time clock and used to provide a time reference when the control system scan and read two encoder counters. In each control cycle, controller should sample link position data from two encoder counters at least one time. In order to provide an accurate time value, timer counter should have 1ms or less time resolution.

3.2.4. Data Transmit Rate

Basically, in one reading cycle, the computer will fetch 6 bytes data from two encoder counters and 2 bytes data from timer counter at the same time. In one writing cycle, computer will write a 12-bit data to the D/A converter. As mentioned before, one

control cycle has less than 10ms and consists of one reading cycle and one writing cycle. Assuming every reading/writing cycle takes 5ms of maximum time, data transmit rate of communication interface between computer and controller interface board should be faster than 8byte*8/0.005=12800bps.

3.3. Desired Features

The following criterion will meet in this project

1) Easy to install

   Ensuring the device is easy to use. No additional card needs to be installed in computer. Computer will serves as a pure digital controller to process control algorithm and use common port to communicate with external hardware. No more setting needs for the hardware.

2) Small in size

   In this project, electronic components are integrated on one controller interface board. It can simplify the interface and reduces the noise interrupt.

3) Expandable interface with computer

   Other device will be able to communicate simultaneously with the PC. This will avoid the device disturb the control system and makes system to be easy to add more devices to increase its function.

4) Inexpensive

   Remove all unnecessary hardware configurations from previous design. A new controller interface board should be much cheaper than previous one.

5) No restriction

The hardware should be supported Windows98/2000/XP. A simple human interface is provided in control software. Output data can be shared with other application software.

Chapter 4

Proposed Design and Option Discussion

4.1. Introduction

The idea of current design is attempting to integrate all hardware into one controller interface board. The controller interface board has multiple functions, such as D/A converter, feedback encoder counter and real time counter, which are independent from the PC.

4.2. Functional Overview

The system flow chart is given in Figure 4.1.

In this diagram, all of the peripherals on the board are controlled by a central microprocessor. The PC is only used to handle the complex algorithm calculation, human interface and the data process. Between the PC and the controller interface board, they exchange data bytes in a special format through a universal serial communication interface. The purpose of doing so is to separate the controller interface board from PC. Any external hardware control will not require PC to handle by adding extra interrupts. It will let PC run in full speed and reduce any possibility of disturbance from outside. Between the USB interface card and microprocessor, a parallel I/O bus is used for writing data from USB interface card to microprocessor and a I^2C bus is used for reading data array from microprocessor to USB interface card. The encoder counters and D/A converter interface with microprocessor with 8-bit data bus. D/A output is an analog signal for driving Pendubot motor through a servo amplifier. In the system, the output signals of the links optical encoder are system feedback signal read by encoder counters.

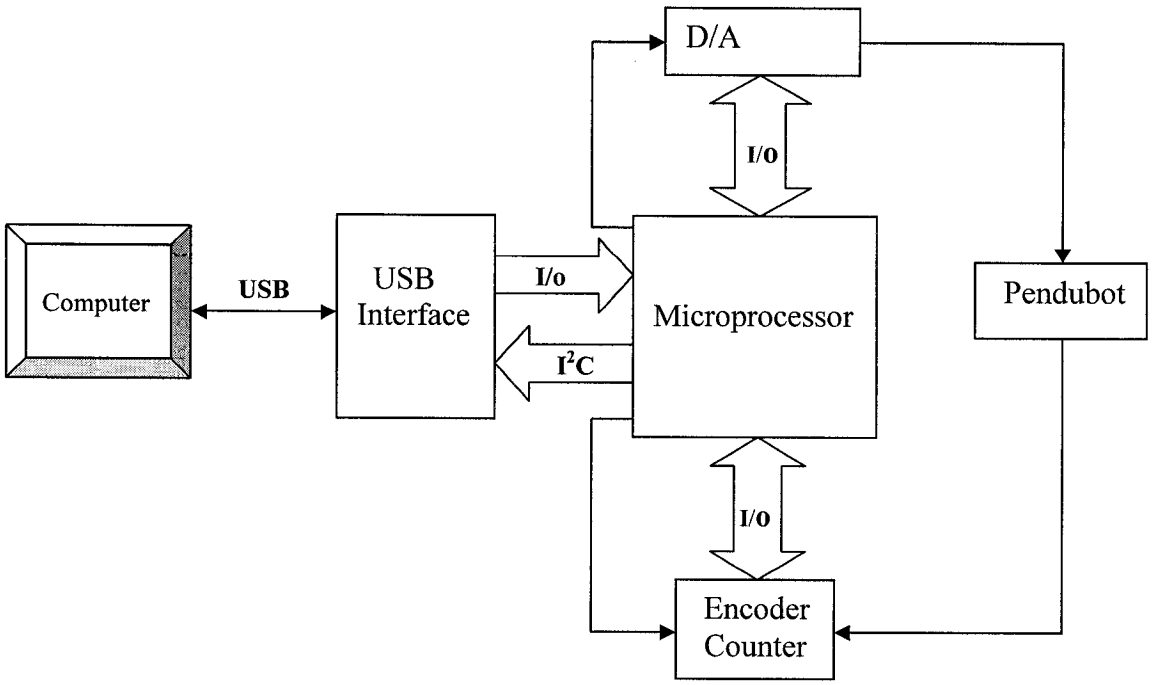

Figure 4.1 System flow chart diagram

4.3. Communication with PC

4.3.1. Serial Communication Options

Several communication options were initially investigated. They are RS232, RS485, low speed USB1.1 and full speed USB1.1.

A RS232 is a serial communication interface port, which allows asynchronous communication between devices. All computers have one or two RS232 ports, called computer serial port for mouse or any device needs to communication with PC. It can work in full duplex operation and transmit data at normal speed of 9600 bps bit rate. RS232 is the most popular computer interface used now. There are plentiful development tools and examples for this interface. But RS232 is normally used in small volumes, low

transfer rate communication between the computer and devices with 9600bps bit rate. This bit rate is not enough for this application.

RS485 is kind of improved RS232 and uses a different principle for multipoint communications. RS485 uses one twisted pair (TP) line, which is two wires twisted around them, for data signal transmission. For RS485 interface, its cable can be up to 1200 meters (4000 feet) long, and commonly available circuits work at 2.5 Mbps transfer rate.

Because for the RS485 only one device would be able to communicate with computer at a time, this would lead to complex timing conflict when multiple devices are using at same time. It is also not a standard communication interface for most computers and an extra hardware and a driver are required if user want to use this port for interfacing with computer. Therefore, RS485 is also not a good option for our design.

The universal serial bus (USB) interface has become more and more popular for transferring data between PC and any available peripherals. It combines all of the advantages of a multi-platform standard—including decreased cost, increased compatibility with a high data transfer rate. It allows the user to simply plug in the device and it is ready to use.

Now there are two revisions of USB interface, which are USB 2.0 and USB1.1. USB 2.0 is kind of hi-speed external bus that support data rates up to 480Mbps. But USB 2.0 has a very complicated protocol and needs special and very expensive tools to develop it. In additional, USB 2.0 has not become a standard configuration in today's computer. An extra hardware card is required if a device need USB 2.0 port to communicate with PC.

USB 1.1 is a relative low speed rate interface comparing with USB2.0. It also has two speeds: one is low speed USB1.1, which support data rate 1.5Mbps; another is full speed USB1.1, which can support up to 12Mbps. For these two USB 1.1 interfaces, they all can satisfy the data transfer speed and would be the best data communication option for this design [23].

4.3.2. Verify Low Speed USB 1.1 Data Transmit Rate

Although both of low and full speed USB 1.1 can match data transmit rate in this design, it is still necessary to make a verification test before any design start because the control system response speed is the most important for the control system and it mainly depends on data transmission rate.

Different microprocessor with built in USB feature was considered. Most of these kinds of microprocessors available in the market are low speed data rate (1.5Mbps). Only few are full speed USB1.1 (12Mbps). Microchip provides two versions of microprocessor in series 16xx (OTP) and 18xx (flash). Both of them are low speed data rate USB1.1. Motorola also has some new low cost 8-bit flash memory microprocessors available in the market, which just support USB low speed rate. They are MC68HC908JB8 and MC68HC908JB16. The MC68HC908JB8 has a USB1.1 low-speed module on chip with 1.5Mbps data rate. The MC68HC908JB16 has a USB 2.0 low-speed module on chip with 1.5Mbps data rate. The EZ-USB AN2131QC supports full-speed data rate (12Mbps), which is 8-bit, 8051 core Cypress's product.

After investigation, the MC68HC908JB8 became our first choice. The advantages to choose this microprocessor are:

1) MC68HC908JB8 is a low cost, high performance USB microprocessor.

2) Motorola provides all free development tools and Windows driver.

3) There are many documents and design examples for reference.

The disadvantages are:

1) We have to develop all USB protocol firmware for this microprocessor;

2) Because the micro has not enough I/O ports, an I/O expansion circuit has to be added in.

4.3.3. Test Method and Test Circuit.

First of all, a USB protocol firmware was developed for MC68HC908JB8. It is written by assembly code. At PC side, application software was also developed in C++ by using Borland C++ Builder with a simple Window interface. Because no software bus protocol analyzer can be used and the complicated USB enumeration process, debugging the newly developed firmware costs much time to pass through. Fortunately, an USB communication was successfully built up between host computer and microprocessor.

All data transfers through USB bus are initiated with the host controller. Each USB transaction is comprised of a serial of packets and mostly involves three types of packets i.e. token packet, data packet and handshake packet. Only data packet contains real data segment that will be exchanged between host and device; other packets are used for configuration or acknowledge. A data packet allows up to the maximum size of 64 bytes data. Therefore, the real data transfer only takes part of the bandwidths of total 1.2Mbps of data rate. Although the 1.2Mbps data rate of low speed USB 1.1 is faster than the required data transfer rate in this project, a test is still necessary.

For each USB transaction, data cycle period is another important factor to affect control system response time. This time period is also measured in the test.

A testing circuit diagram for USB 1.1 low speed is given in Figure 4.2.

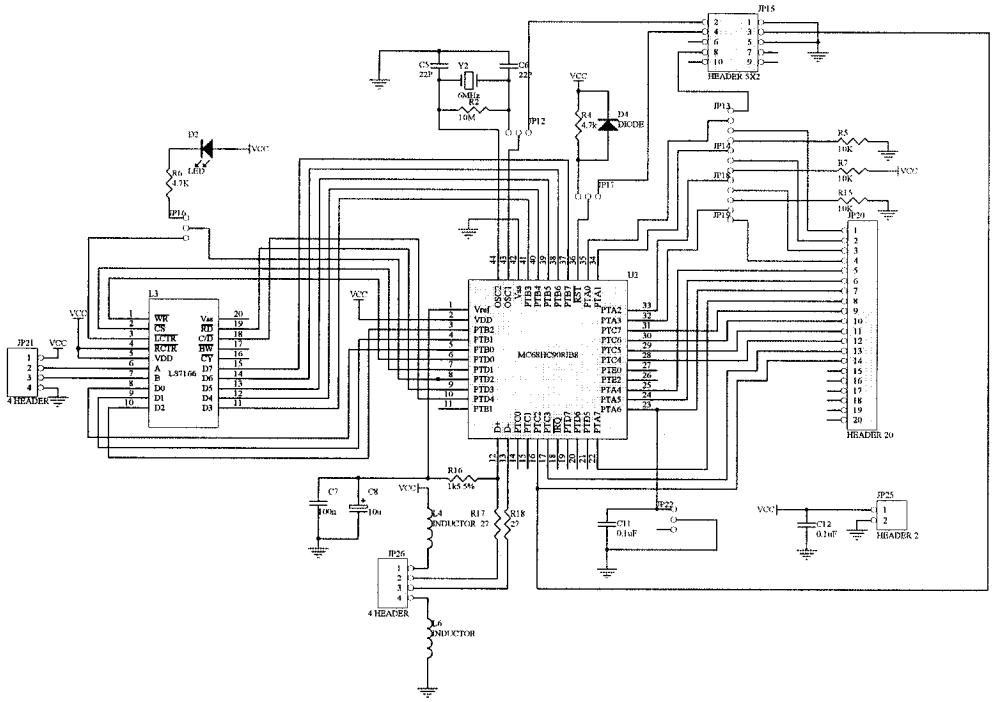

Figure 4.2 Test diagram for low speed USB 1.1

In the figure 4.2, JP15 is in-circuit programming port. JP13, JP14, JP18, JP19 are jumpers for setting microprocessor in either programming mode or normal mode. JP23 is the USB connector.

In user program, the USB interrupt is actually triggered by calling getUSB() function. In order to measure data cycle time, a infinite while loop is used to call getUSB() function twice. The first getUSB() function gets a byte=0x00 and set MC68HC908JB8 PTA0 of PORTA low, then the second getUSB() function gets a byte=0x01 and set PTA0 high after. By measuring output of pin PTA0 with an oscilloscope, a square wave can be

obtained. The period of the square wave is the time taken for calling two getUSB() functions. The pseudo code of testing USB1.1 low speed is given below:

```
for(;;) {
        do {    // get data to output pipe
                io_buffer[n++] = getUSB(); // get the data array from host
                } while(n<8);                   //the data packet has 8 bytes
        if(io_buffer[0]==0)                     //if first byte of array =0
                set PTA0 low
        else  set PTA0 high
        do {
                io_buffer[n++] = getUSB();
                } while(n<8)
        if(io_buffer[0]==1)   //if first byte of array =1
                set PTA0 high
        else set PTA0 low
}
```

In the testing program, a data array was written from PC to microprocessor in each USB transaction. For this design, 8 bytes data is required to be transferred in each read/write cycle. When the data array is changed from 2 bytes to 16 bytes, the testing result shows that it does not change the output frequency of the square wave. Therefore, the data transfer rate of slow speed USB1.1 is fast enough.

The real measuring result indicated that the interval time between two USB transactions is about 40ms. In this design, a 10ms or less interval time is required. So the USB low speed interface could not satisfy for this control system.

4.3.4. USB 1.1 Full Speed Interface

The full speed USB 1.1 has 10 times faster than low speed USB 1.1, which can support up to 12Mbps data rate. But in the 8-bit microprocessor market, only a few companies provide this kind of version product. To develop its protocol firmware and windows driver are another problem, because it not only needs to fully understand its complicated protocol but also needs professional tools.

Fortunately, the Desasys Company provides a ready-made USB I2CIO interface product. This USB interface board uses a Cypress AN2131 EZUSB microprocessor, which has on-chip USB 1.1 full speed port. The board provides a 20-bit of user-configuration digital I/O, and a fast I^2C interface with embedded protocol firmware and necessary Windows device driver. It is a very convenient development environment because it allows loading the designed firmware into the EZUSB microprocessor's RAM through the USB bus.

The Windows API calls allow initializing a board, configuring its I/O pins for input or output, reading and writing pin values. So it is a very good USB interface solution and makes development fast and easy. The trade-off is another microprocessor has to be used in the design to serve peripherals control.

4.3.5. USBI2CIO Board Test

Because full speed USB1.1 has a faster data rate than the low speed USB 1.1, its data transfer speed will not have the problem for this design. But the data transfer interval time of full speed USB 1.1 still need to be verified to decide if the board can be used in the design. In Windows application software, the function ReadIoPort and WriteIoPorts can be called to read and write I/O port on USBI2CIO board. A similar testing pseudo code is given below:

```
        hDecInstance=INVALID_HANDLE_VALUE // Open device
While{      //a infinite loop for continue writing data to the board
        WriteIoPorts( 0x00000000);  // write hex code 0x00000000 to I/O port
        WriteIoPorts( 0x00000001);  // write hex code 0x00000001 to I/O port
        }
```

By reading the square wave output from I/O port, interval time can be obtained. In the testing, each read/writes cycle took about 4 ms, it works out to 250 reads per second. This is plenty fast for this design.

4.4. D/A Converter

There are many D/A ICs available in the market. In this design, a 12-bit D/A converter is required. The D/A converter should have a very good interface in order to communicate with microprocessor easily. It is better to have an internal voltage reference itself. It will much simplify the D/A circuit design. Finally, a faster digit to analog convert speed will be the most important for a D/A converter.

A Maxim (MX7548) D/A converter IC is chosen for this design. It is 12-bit CMOS, current output, multiplying digital-to-analog converter with a versatile interface for microprocessor using 8-bit buses. It has a very fast interface timing speed: 120ns minimum write pulse width. MX7548 can be easy configured as a bipolar operation.

4.5. Power Supply

There are three kind of voltages are required in the design. The standard IC power supply voltage is the 5 VDC. D/A converter circuit needs +12VDC and –12VDC for current amplifier.

4.6. Microprocessor

There are three major components on controller interface board need to be controlled by a central microprocessor, which include two optical encoder counters and one D/A converter. The microprocessor also acts as an interface controller in order to communicate with a USB interface card. These required that a microprocessor have four 8-bit bi-direction I/O ports and at least 4K internal RAM memory size. And internal bus frequency of microprocessor should be faster than 4MHz.

MC68HC908MR32 is an 8-bit general-purpose microprocessor from Motorola. It has six I/O ports up to 44 input/output pins with 32K flash memories. Its internal bus frequency is programmable in a wide range of multiplying external crystal frequency. MC68HC908MR32 has in-circuit programming feature; no traditional programmer unit is required to erase or update its flash memory. Another main reason to choose this

microprocessor is that its emulator and full function C compiler are all available in our lab.

4.7. Encoder Counter

Using microprocessor to read the optical encoder output signal directly is possible. Although it needs to develop an additional firmware for microprocessor to do it, it could simplify hardware design and use fewer components. But in order to read encoder signal, the microprocessor needs to have external interrupt feature. For two optical encoders, totally four external interrupt pins are required. So far it is hard to find this kind of microprocessor and it will lead to complex timing conflict when it is running in very high frequency.

LS7166 is a professional high-speed 24-bit encoder counter. It can be programmed to operate in different mode, such as up/down, binary, 24 hours clock, etc. LS7166 will enable a microprocessor to track the speed, direction and position of an optical incremental shaft encoder. An 8-bit bus allows LS7166 to connect with any microprocessor easily and up to 10 MHz signal can be read from LS7166.

Chapter 5

Hardware and Software Design

5.1. D/A Converter Circuit Description

5.1.1. Features

The main component in D/A converter circuit is the MX7548. The basic principle of MX7548 DAC circuit consists of a laser trimming, thin-file R-2R resistor arrays with NMOS current switches as shown in Figure 5.1 [21]. Binary weighted switches to either OUT1 or AGND depending on the status of each input bit.

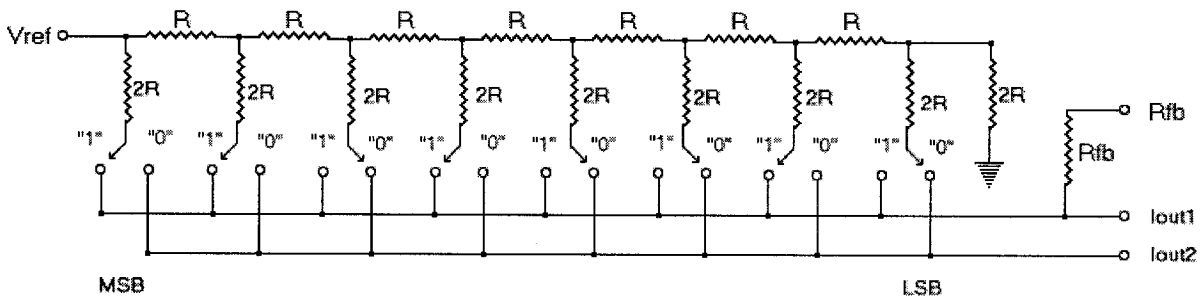

Figure 5.1 R-2R resistor arrays

The current output can be converted into a voltage signal by adding an external output amplifier.

The amplifier circuit diagram is shown in Figure 5.2.

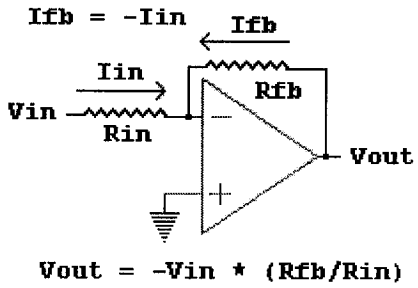

Figure 5.2 Amplifier circuit

I_{out1} of the R-2R ladder is connected to the inverting input of the Op-Amp. I_{out2} of the R-2R ladder is connected to the non-inverting input and to ground. One end of the internal R_{fb} feedback resistor is connected to the output of the external Op-Amp. The other end is internally connected to the R-2R ladder's I_{out1} as shown above. Thus, it is connected from the output of the Op-Amp to the inverting input.

Since the currents are equal but opposite, they cancel each other out, resulting in 0 volts at the inverting input. Thus, the inverting input is at the same potential as ground. This generated ground equivalent is termed a virtual ground.

Because the MX7548 is needed to configure a bipolar operation for an output voltage range from -10V to +10V, a second amplifier and three matched resistors, R3, R4 and R5 are required. These resistors must be made of the same material (preferably metal file or wire-wound) for good temperature characteristics, and they should match to 0.01% or better for 12-bits performance.

The whole DAC circuit is given in Figure 5.3.

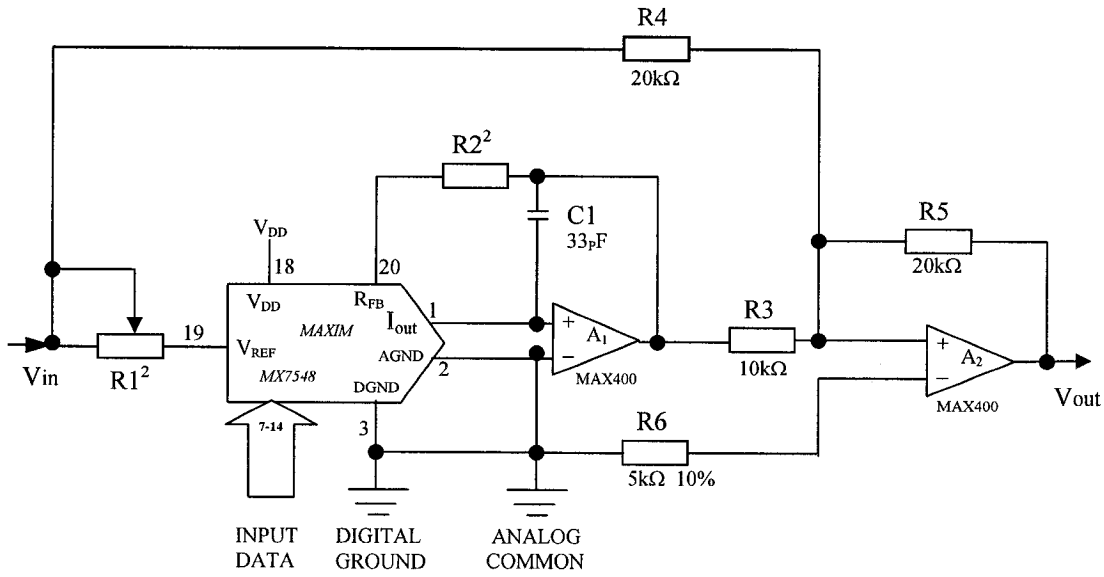

Figure 5.3 D/A converter circuit

The code table for the output of D/A converter is listed in Table 5.1. In multiplying applications, the MSB determines output polarity while the other 11-bit control amplitude.

DIGITAL INPUT MSB LSB	ANALOG OUTPUT
1111 1111 1111	$+V_{REF}\left(\frac{2047}{2048}\right)$
1000 0000 0001	$+V_{REF}\left(\frac{1}{2048}\right)$
1000 0000 0000	0
0111 1111 1111	$-V_{REF}\left(\frac{1}{2048}\right)$
0000 0000 0000	$-V_{REF}\left(\frac{2048}{2048}\right)$

Table 5.1 Output binary code table

R1 is an adjustable resistor. By trimming R1, while the DAC with a code of 1000 0000 0000, the output voltage should be 0V.

DACs are typically very fast, limited only by the switch speeds and the slew rate of the OP-amp. Comparing with other devices in the project, the D/A converting time can be ignored.

5.1.2. Interface with Microprocessor

Following is the MX7548 pin assignments:

DF/DOR (pin 5) – Data format/data override

CTRL (pin 6) – Control input

These two pins should be used together to control data format and override. Here they both are set high for Right-justified data format.

CSMSB (pin 4) - chip select most significant byte. It is active low for loading MS byte input register.

CSLSB (pin 16) – chip select least significant byte; active low input.

WR (pin 17) – write input; active low input.

LDAC (pin25) – load DAC input; active low.

The pins of WR, CSMSB, CSLSB and LDAC should be used in combination with each other for loading 12-bit input data into DAC register. The combination cases are list in Table 5.2.

$\overline{WR}$	$\overline{CSMSB}$	$\overline{CSLSB}$	$\overline{LDAC}$	Function
0	1	0	1	Load LS Byte into Input Register
0	1	0	0	Load LS Byte into Input and DAC Registers
0	0	1	1	Load MS Byte into Input Register
0	0	1	0	Load MS Byte into Input and DAC Registers
0	1	1	0	Load Input Register Contents into DAC Register
1	X	X	X	No data transfer

Table5. 2. Data Load and Transfer Control Table

MX7548 can run in either automatic transfer mode or strobes transfer mode. Because MX7548 is running alone in the system, automatic mode is selected. The operation-timing diagram for automatic mode is given in Figure 5.4.

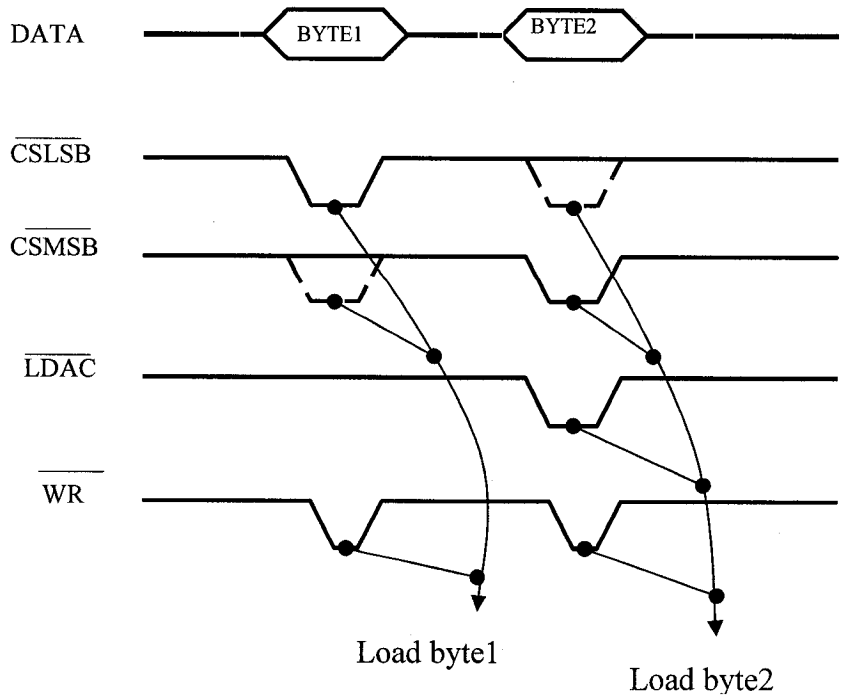

Figure5.4 Writing timing diagram for auto mode

Figure 5.5 shows MX7548 interface circuit with MC68HC908MR32 microprocessor. Here they used an 8-bit data bus for data communication. To load a 12-bit data word to MX7548, microprocessor will use two writing cycle to complete it.

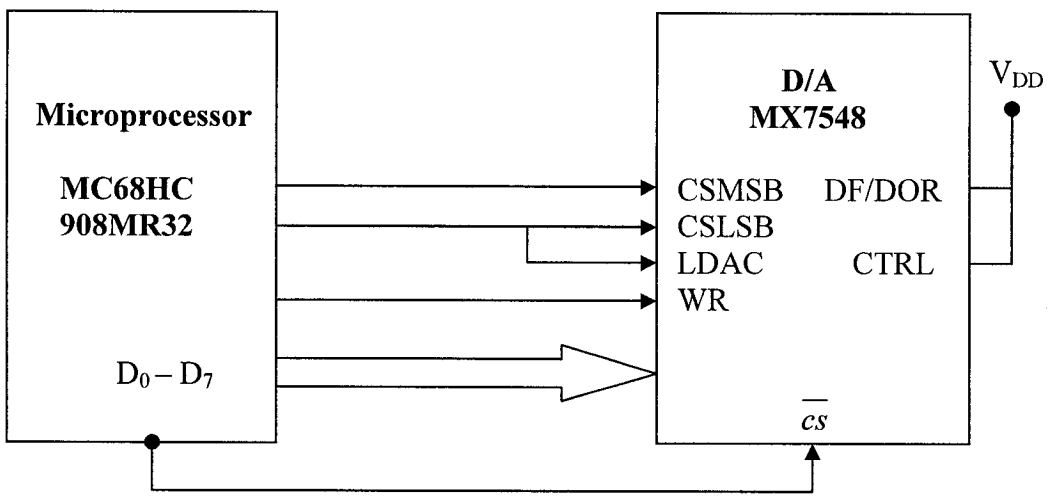

Figure 5.5 MX7548 interface circuit to MC68HC908MR32 microprocessor.

5.1.3. Software Design

The pseudo code to fetch and load the 12-bit data word to MX7548 is given below:

```
DAC subroutine: // the sub routine is called by timer interrupt in constant interval.
Write_DA_converter( )        //read DA subroutine
{
        low CSLSB pin
        low WRITE pin
        write D/A LSB data
        high WRITE pin
        high CSLSB pin

        low CSMSB pin
        low LDAC pin
        low  WRITE pin
        write D/A MSB data
        high WRITE pin
        high  LDAC pin
        high CSMSB pin
}
```

5.2. LS7166-Optical Encoder Counter

5.2.1. Feature

A CMOS 24-bit quadrature counter LS7166 chip, which is made by CSI (Computer System Inc.), is used to count the pulses of two optical encoders [17]. The chip can be programmed to operate in many modes. In this project, Up/Down count mode

is selected. The IC communicates with external circuit through 8-bit data bus. Control and data words are written or read through this 8-bit three states I/O bus.

There are 6 special registers in LS7166. They are:

1) PR-Preset register is the input port for the counter. Using this register can preset counter value;
2) MCR-"master control register". Perform register reset and load operations
3) ICR-"input control register". Initializes counter input operation modes.
4) OSR-"output status register". Indicate counter status.
5) OCCR-"output control register". Initializes counter and output operating modes.
6) QR-"quadrature register". Selects quadrature count mode.

Because a 24-bit data value of the counter can only be accessed through output latch (OL), the data needs to be latched into the OL under software control by setting bit 1 in counter MCR register to "1". After that the data can be read through the CNTD. The transfer is least significant byte first, most significant byte last.

Pin 8--Pin 15 (D0-D7) are data bus pins for interfacing with the system bus; Pin 2 ($\overline{cs}$) is chip select input pin. A logical "0" at this pin enable the chip for read or write; Pin 19 (RD) is read pin, A logical "0" at this pin enable the OSR and the OL to be read on data bus; Pin 1 (WR) is write input pin. A logical "0" at this pin enable the data bus to be written into the control data and data register; Pin 18 (C/D) is control/data input pin. A logical "1" at this pin enable a control word to be written into one of four control registers or the OSR to be read on the I/O bus. A logical "0" at this pin enable data word to be written into the PR, or the OL to be read on the I/O bus; Pin 6 (A) is the up count

input; Pin 7 (B) is the down count input; Figure 5.6 give the configuration circuit for LS7166.

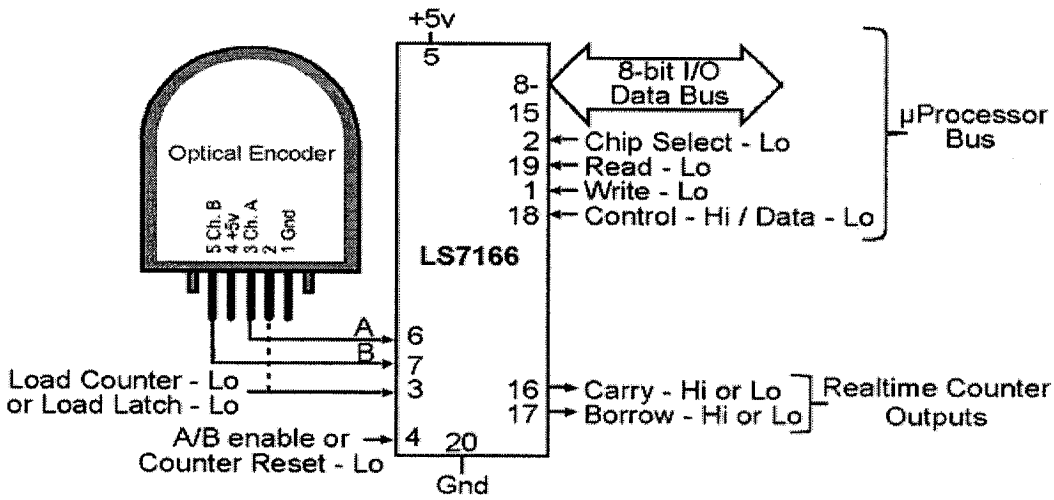

Figure 5.6 LS1766 configuration circuit

The incremental encoder used in the project has two output channels (A and B) to sense link position. Using two code tracks with section positions 90° out of phase (Fig5.7), the two output channels of the quadrature encoder indicate both position and direction of rotation. If A leads B, the shaft of encoder is rotating in clockwise direction; If B leads A, then the shaft is rotating in a counter-clockwise direction.

The quadrature counter LS7166 is used to sample the output signals of channel A and B. Based on the past and present binary states, LS7166 outputs a count signal to the internal counter.

By decoding quadrature clock input channel A and B, LS 7166 can derive out three different kinds of frequency clocks for internal counter. They are called X1, X2 and X4 clock. X1 has lowest resolution and X4 has highest resolution (Figure 5.7).

Generally, the count will increment when the shaft rotating one direction and decrement in the other direction.

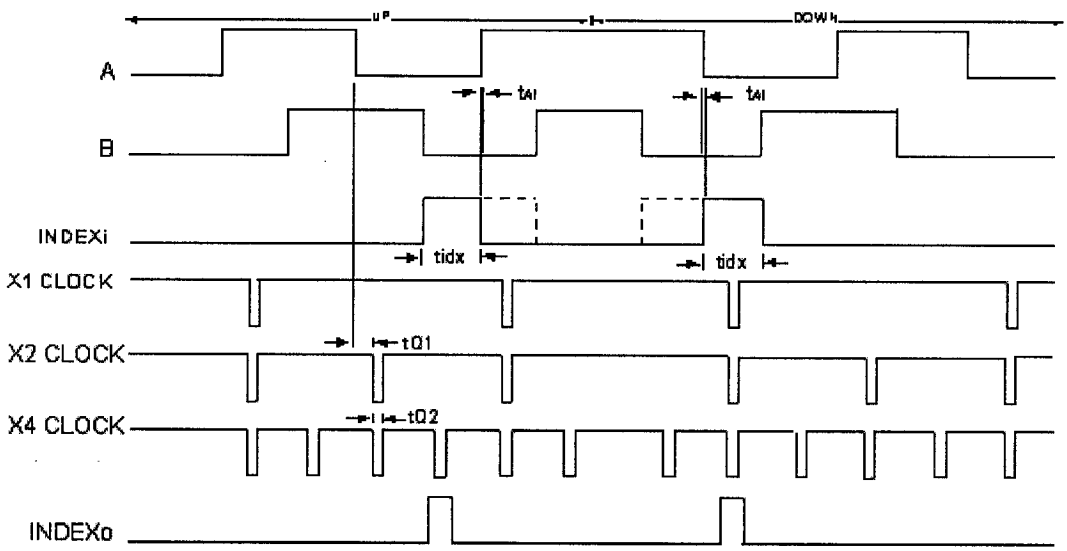

Figure 5.7 Signal A, B timing diagram

The LS7166 counter chip required some basic initialization, before it is able to decode the quadrature counting pulse of the incremental encoder. To do this the following data sequence has to be load into the CNTCS1-Counter control and statues register.

1) Write 0x20 to CNTCS1-select the counter MCR register, perform a master reset on the LS7166 counter chip.
2) Write 0x68 to CNTCS1- select the counter ICR register; enable data input A/B; configure pin3 as counter load input (required for the reference logic).
3) Write 0xB0 to CNTCS1-select the counter OCCR register, configure pin16 as compare result output (required for generation of interrupts at defined counter values).
4) Write 0xc1, 0xc2 or 0xc3 to CNTCS1-select the counter QR register, enable quadrature mode x1, x2, and x3.

After this initialization the counter will begin decoding the encoder quadrature signal.

For reading LS7166 24-bit up/down counter, microprocessor needs three reading cycles in sequence because LS7166 only has an 8-bit data bus for data communication with microprocessor. Reading procedures are given below:

1) Write 0x03 to CNTCS1- select the counter MCR register, latch the actual counter data value into the output latch (OL), and reset the OL address pointer;
2) Read CNTDA- read data byte0 (LSB), increment OL address pointer;
3) Read CNTDA- read data byte1 (LSB), increment OL address pointer;
4) Read CNTDA- read data byte2 (LSB), increment OL address pointer;

Read cycle timing diagrams:

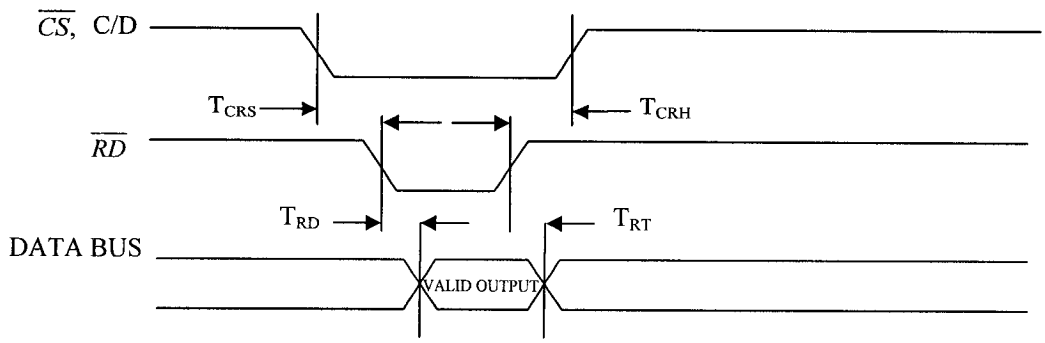

Figure 5.8 Read cycle timing diagram

And write cycle timing diagram:

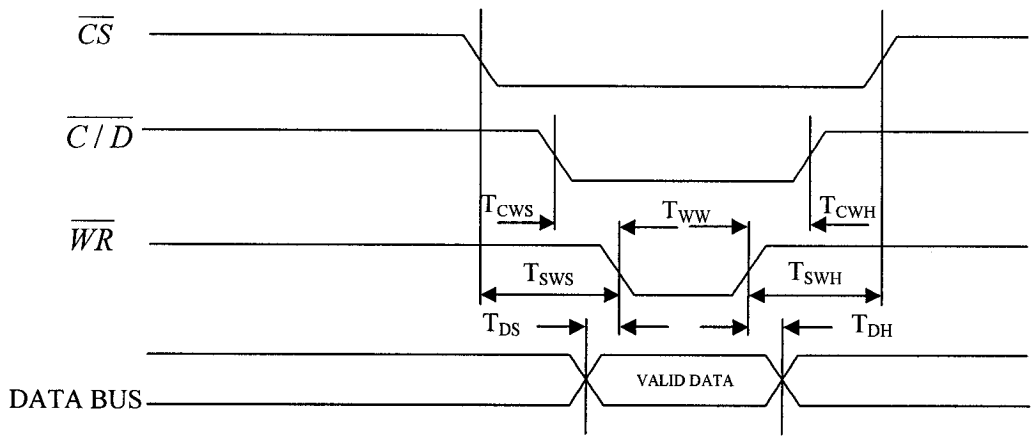

Figure 5.9 Write cycle timing diagram

5.2.2. Circuit Diagram

In this project, two LS7166 ICs are used. They share an 8-bit data bus and are connected to microprocessor general I/O port A. IC is a 26LS32A, R1 to R4 are balance resisters. R5 to R13 are pulling up resistors for encoder input lines. Two 9-pin serial gives the input circuit of LS7166.

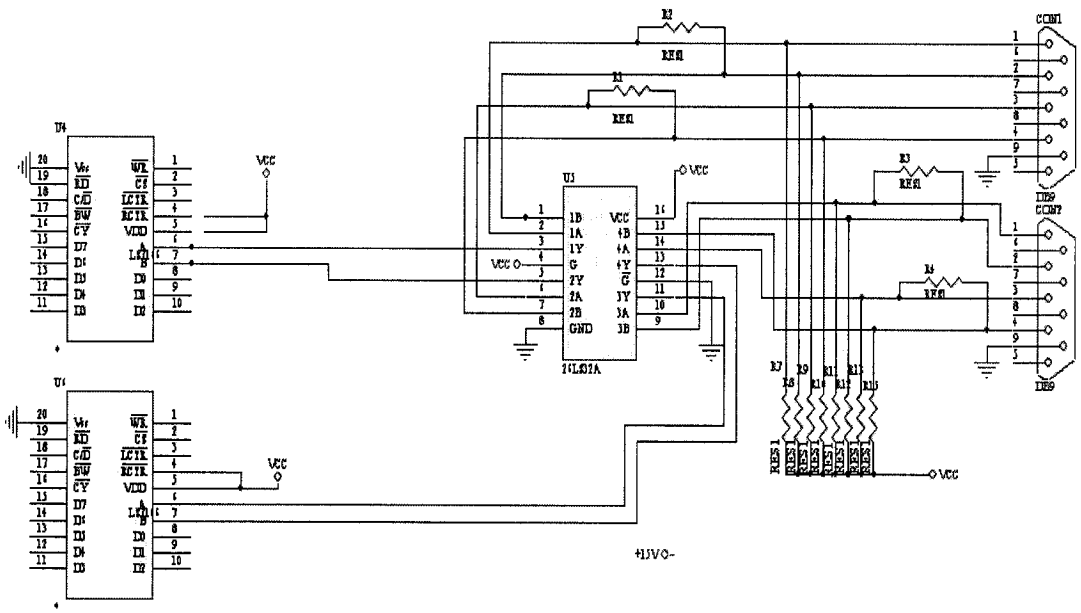

Figure 5.10 LS7166 input circuit

There is an IC (26LS32A) between encoder and quadrature counter [9]. The 26LS32A is a quadruple differential line receiver for balanced and unbalanced digital transmission. The purposes of using this receiver are:

1) It could improve system sensitivity because there is an additional stage of amplification in 26LS32A.

2) Act as a input buffer to prevent any unexpected voltage spike or big current to damage system

3) Although an unbalanced signal is used in this project, 26LS32A allows the system to read a balanced signed too.

4) Increased input impedance.

5.3. Power Supply

Figure 5.11 shows 0power supply circuit design. The power supply unit uses 3 fixed-voltages 3-terminal regulators, which allow a wide range of input voltage to be used. Referring to the Figure 5.11, an LM7805 regulator provides the stabilized +5 VDC for the microprocessor and peripheral devices. LM7912 and LM7812 provide the stabilized +12VDC and –12VDC for D/A Op-Amps. Figure 5.11 gives the design circuit diagram. In the diagram, C1 provides decoupling and smoothing of the unregulated DC supply. The parallel combination of C2, C3 provide high and low frequency decoupling to the +5 VDC. C5, C6 provide similar function for +12VDC and –12VDC.

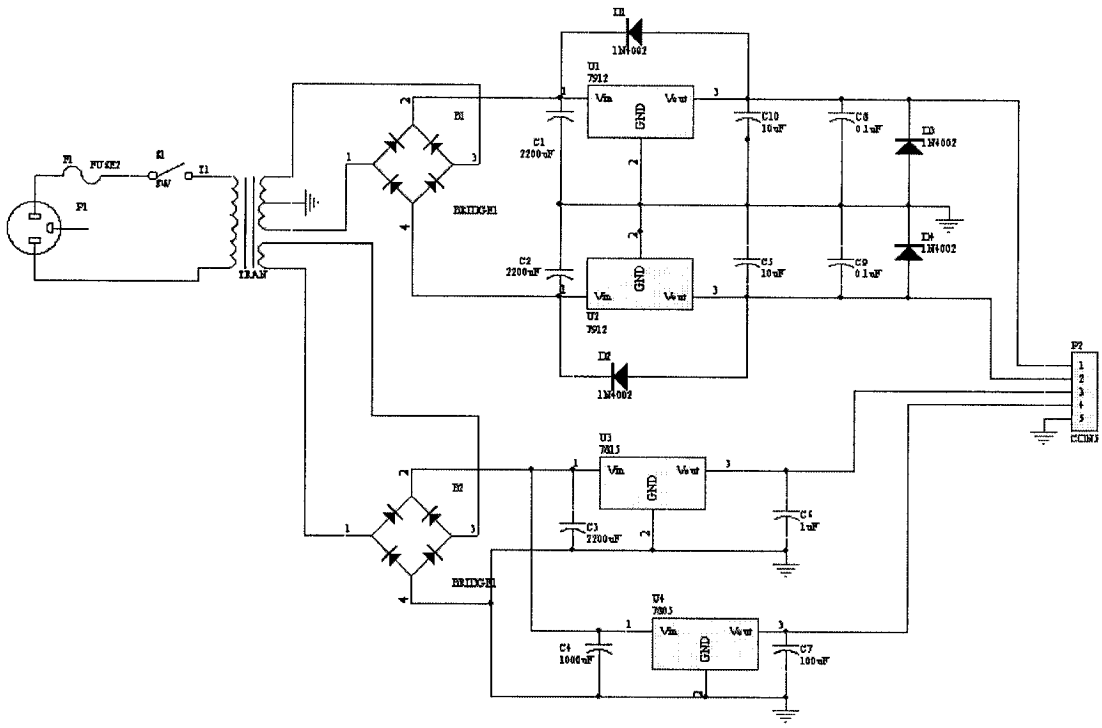

Figure 5.11 Power supply circuit

5.4. Microprocessor

5.4.1. Feature

The central control unit is an 8-bit Motorola microprocessor (MC68HC908MR32) [18]. The MR32 is a new member of the low-cost, high-performance MC68HC08 (HC08) family of 8-bit microprocessors, designed specifically for mid-range control applications. MR32 has 32k on-chip flash memories with in-circuit programming feature. Its internal bus can run high up to 8MHz, which provides fast enough speed to meet system requirement. In its 64 pins packet, it provides 37 general-purpose input/output ports and 7

input only ports. It also has two 16-bit counter timers that can operate as free-running counter or a module up counter.

5.4.2. Hardware Configuration

MR32 configuration circuit is given in figure 5.12. In this figure, X1, C1, C2 and R1 consist of a Pierce oscillator configuration circuit with internal amplifier. X1 is a crystal oscillator, which frequency is 4.1952 MHz. This frequency is special for in circuit programming purpose. Because inside MR32 there is a clock generation module with phase-locked loop circuit (PLL), a highest 7.3416 MHz bus frequency can be generated by setting frequency multiplier factor to 7 in PLL programming register. C1 and C2 are load capacitors for phase shift. R1 is feedback resistor. For PLL circuit, C3 and C4 are two external capacitors. The C3 is the bypass capacitor and C4 is noise filter capacitor. V_{dd} and V_{ss} are power supply and ground pins. The MCU operates from a single power supply. To prevent noise problems, a bypass capacitor is placed before power supply. C8 is a high –frequency –response ceramic capacitor, which should be located as close to the microprocessor as possible.

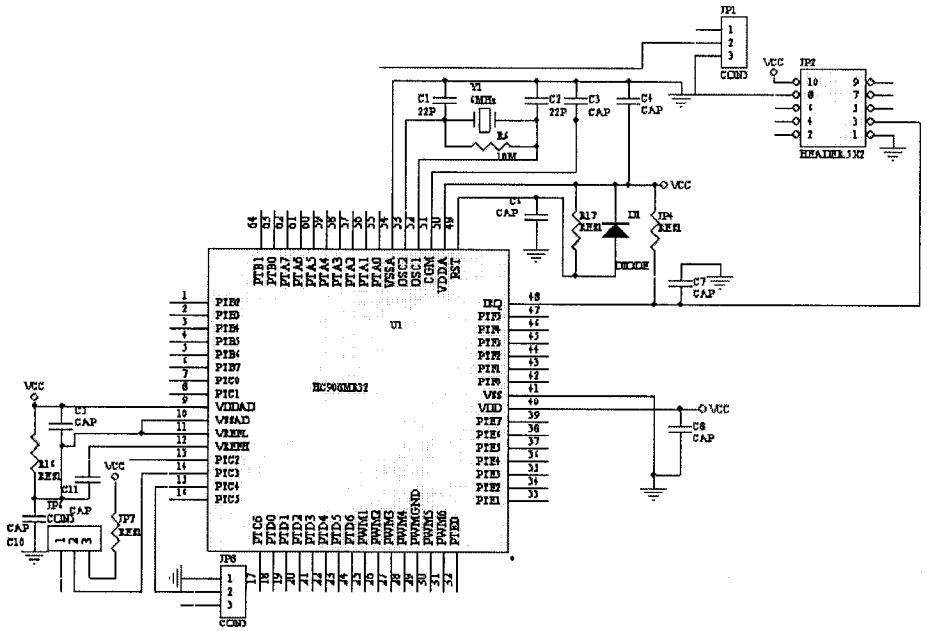

Figure 5.12 MR32 configuration circuit

IRQ is external interrupt pin, which is used for detecting the start signal of I^2C in this project and is pulled high (5V) by resistor R in normal running case. It also can be linked to pin 3 by changing the jumper of JP2 for programming purpose.

R17 and D1 consist of a simple reset circuit for keeping reset pin high; C8 is a bypass filter to protect reset pin from high frequency noise.

V_{redl}, V_{refh} are internal analog digital converter (ADC) voltage reference pins. V_{ddad} is ADC power supply voltage pin. V_{ssad} is ADC power supply ground pin. Although no ADC function is used in the project, a configuration circuit is still necessary. R16 is a current limited resistor; C3 and C10 are noise bypass filters. For maximum noise

immunity, these two capacitors should be place as close as possible to the package; C11 is an ADC referent voltage decoupling capacitor.

An in-circuit programming connector is added into circuit. It will provide much convenient method to reprogram microprocessor firmware. Motorola designed a very simple way for host computer to access internal memory by assisting of its on-chip firmware. When MR32 is running in monitor mode by setting a high 12V on external interrupt pin, it can communicate with host computer via a standard RS232 interface. In monitor mode, microprocessor can execute host computer code in RAM while all pins retain normal operation mode function. All communication between the host computer and MR32 is through the PTA0 pin while PTC2 and PTC4 need to connect to ground and PTC4 requires a pull-up resistor. This is the reason why there are three jumpers in circuit. These jumpers can be set for micro either operating in normal mode or in programming condition.

PORTA is configured as normal bi-direction I/O port. This port is used as 8–bit data communicating bus between MR32 and two LS7166. PORTF is configured as normal output port, which is connected with LS7166 function pins for chip control the purpose. PORTE is configured as normal output port, which is used for writing data byte to MX7548. Although the pins from PWM1to PWM6 are special PWM output pins, they can be configured as normal output pins for MX7548 function control. PORTB and 4 pins of PORTC are configured as input pins, which consist as an input port connecting with 12-bit data I/O bus for directly reading the data from USB interfaces board. The pins of PTC4 and PTC5 are configured as normal input/output pins. These two pins are

special defined as clock line (SCL) and data line (SDA) for I^2C bus, which are used for writing numerous data to USB interface board through an I^2C communication interface.

5.4.3. Software Description

An assembly code running in MR32 consists of two parts. One is the main continuous loop and another is an external interrupt subroutine. The code flow chart is given in Figure 5.13.

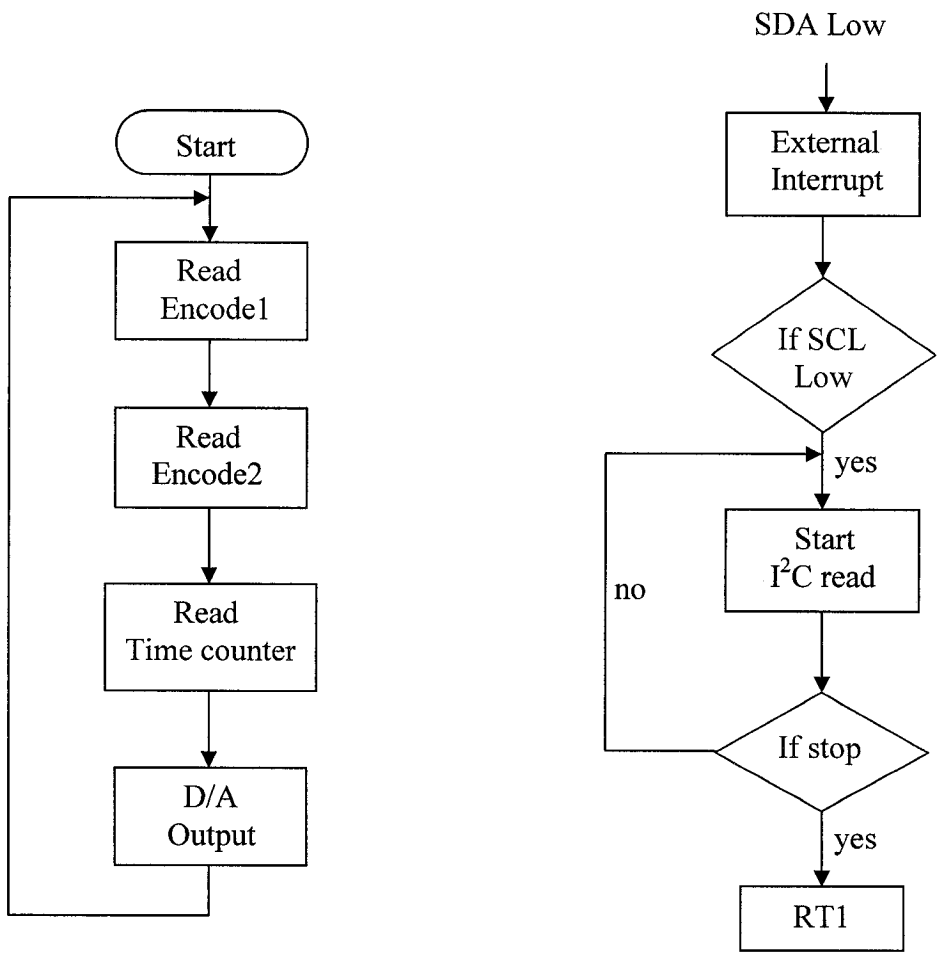

Figure 5.13 The flow chart of control code in MR32

The main function loop will performs following operational sequence:

1) Read two encoder counters. A six bytes data array is updated in the program. Each encoder counter data value takes 3 bytes of the array.
2) At the same time when encoder counters are read, a time value is also recorded by reading a free-run 16-bit timer counter inside MR32 as a time reference.
3) Finally write a 12-bit data to D/A converter.

When the external interrupt pin is pulled to low by master I^2C device, the code jumps into external interrupt subroutine and start to perform the data exchange with USB interface device via I^2C bus. After the interrupt subroutine finishes, the code goes back main loop and continue its task.

5.5. Devasys USBI2CIO Board

5.5.1. I^2C Communication

As mentioned in previous section, there are totally two 24-bit encoder counter data and one 16-bit real time counter are obtained by microprocessor in one running loop. These data need to be transferred to computer through USB interface; at same time there is one 12-bit digital voltage data needs to be received from computer through USB interface as well. According to the specification in chapter three, all the data are required to exchange within 10ms. Because for USBI2CIO board each read/write cycle requires at least 4ms, only one reading cycle and one writing cycle can be used in 10ms. It will be fine to use I/O port to transfer a 12-bit voltage value from USBI2CIO board to

microprocessor because USBI2CIO board has totally 20-bit I/O port; but it will be impossible to transfer a total 64-bit encoder and time counter data from microprocessor to USBI2CIO via parallel I/O [12].

Fortunately USBI2CIO provides a master I^2C interface port for communication with other peripheral; it can transfer up to 64 bytes in one reading or writing cycle. If I^2C port is employed as communication port, the problem can be solved. That requires MR32 have I^2C port as well. Unfortunately the MR32 does not have on-chip hardware model for the I^2C interface.

Because MR32 has high processing speed and flexible I/O port, effective software I^2C slave implementation can be done. MR32 can perform fast (400 Kbps) I^2C slave operation in software. In the specification of USBI2CIO, its I^2C port can reach to the highest 90 KHz speed.

I^2C is a 2-wire communication link requiring a clock line (SCL) and a data line (SDA) to communicate. An I^2C bus has both a master device and a slave device attached to it. A master is defined as device, which initiates a transfer, generates a clock signal (SCL), and terminates the transfer. A slave device is addressed by the master device and replies the master data transfer requirement. The bus also provides some errors checking by using acknowledgment bits during data byte transfer.

By controlling two digital I/O pins, it is possible to simulate I^2C transfer on MR32 with software. These I/O pins should be done by open drain. Because PTC4 and PTC5 are high –density complementary metal oxide semiconductor (CMOS) and not an open drain, some safeguards must be implemented. A series resistor should be connected between the CMOS output pin and receiver's input pin. This will provide some current

limiting. Another consideration is supporting a logic high level for any open-drain receiver pins. A pull-up resistor is used at the receiver's open drain pin to passively pull up to the supply voltage when the pin is not being actively driven low. This pull-up resistor is carefully chosen so that when the master pin drives low, a valid low voltage level is present to the I^2C receiver line. Figure 5.14 is given the connection diagram

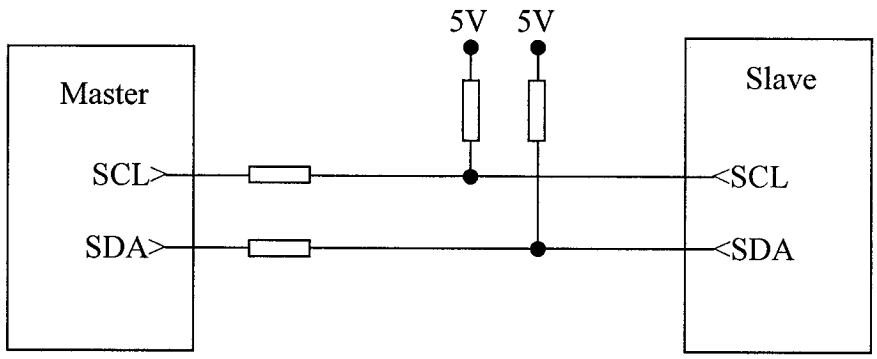

Figure 5.14 The connection diagram

5.5.2. I^2C Protocol Firmware

Data transferring is always initiated by bus master device. A high to low transition on the SDA line while SCL is high is used to be a START condition. A START condition is always following by the (unique) 7-bit slave address and then by a data direction bit. The device addressed now acknowledged to the master by holding SDA low for one clock cycle. If the master does not receiving any acknowledge, the transfer is terminated. Depending of the data direction bit, the master or slave transmits 8-bit of data on SDA line. Then the receiving device acknowledges each success data transfer by holding the SDA line low at ninth clock cycle. A transfer is terminated when the master issues a STOP condition. A STOP condition is defined by a low to high transition on the SDA line while the SCL is high.

Figure 5.15 illustrates the START and STOP condition in I^2C bus.

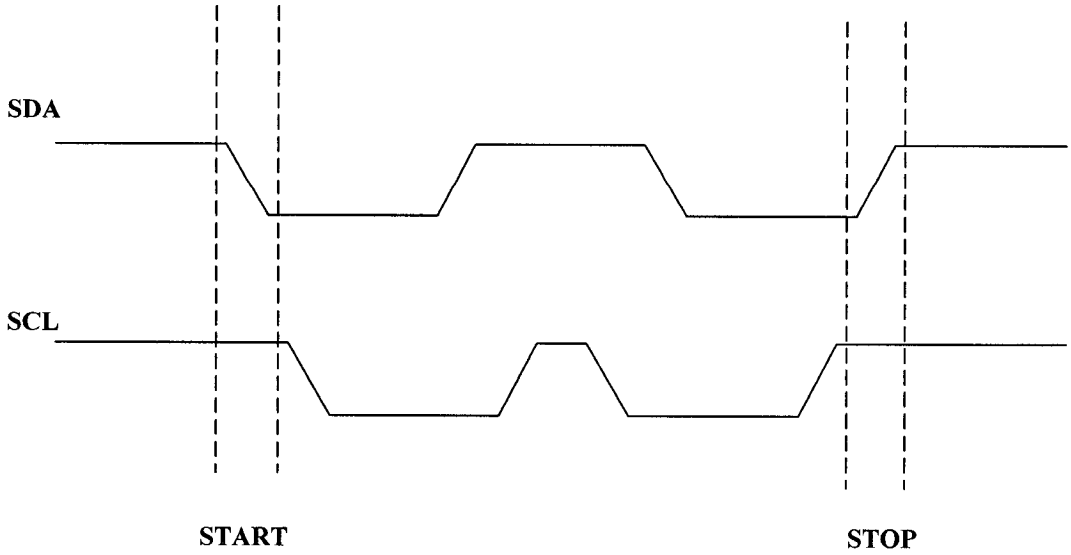

Figure 5.15 START and STOP Conditions on I^2C Bus

Figure 5.16 shows the timing of the acknowledge impulse, which is sent by the slave device after it receives all 8-bit of the transferred byte.

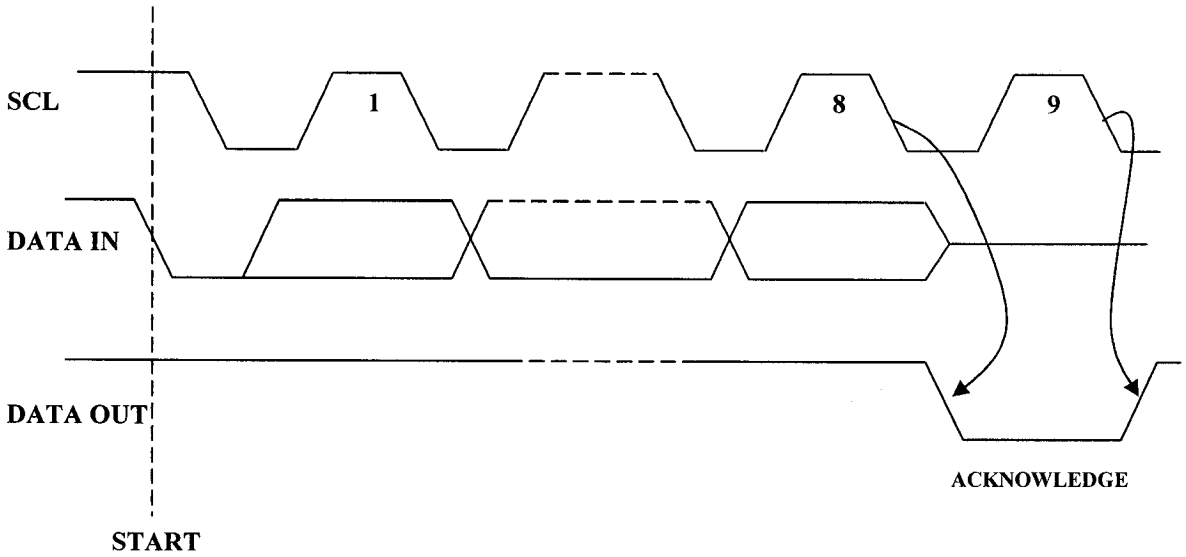

Figure 5.16 Acknowledge Bit Timing on I^2C Bus

The main software flow chart is shown in Figure 5.17. The implemented software makes it possible to use microprocessor as an I^2C slave device in this project. This software routine is divided into two main parts. It is a special initialization sequence executed directly after a reset and the I^2C handle routine.

The initialization routine performs the necessary initialization of PORTC. However if any other devices need to be initialized, this could be done here too.

When initialization is done, the routine enters into a busy loop, which waits for the first START condition. Because a high to low transition on SDA is not necessarily indicated a START condition, if the bus is not free (no activity), both SDA and SCL must be monitored. When a SATRT condition is detected, the routine will disable all system interrupt service. Then slave device starts to receive first 7-bit address and one write/reading bit. If the received address marches with the slave device address, salve device will knowledge master and call either "master_read" subroutine or "master_write" subroutine to start the data exchange transfer until a STOP condition is received. Once the communication is set up, by calling the master_read or master_write subroutine, slave device can received or transfer a byte data once a time. If more data need to be exchanged, which is required from master device, slave device could recall the subroutine and loop here until maximum 64 bytes data. The maximum number of bytes per transfer is defined in USBI2CIO board specification.

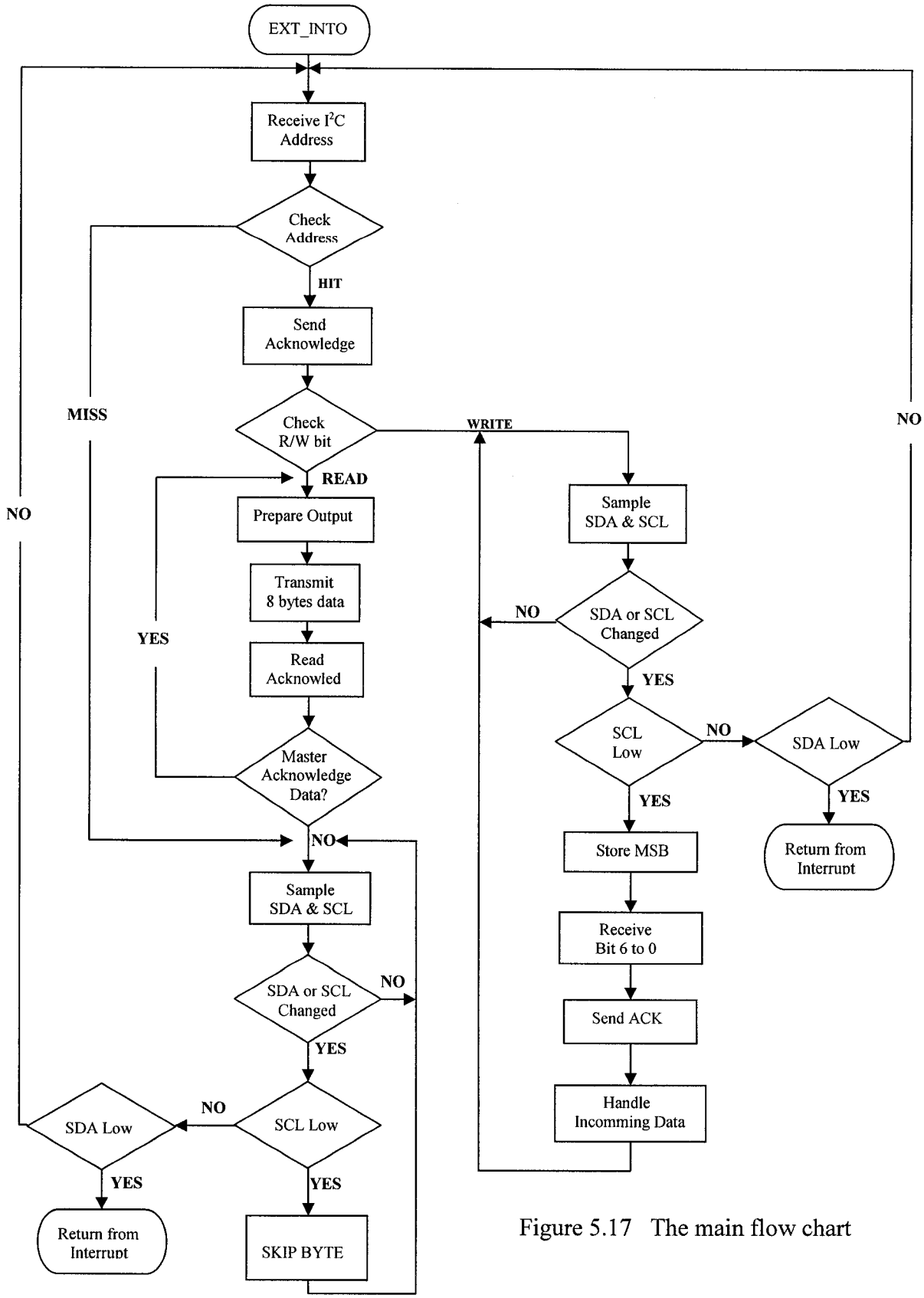

Figure 5.17 The main flow chart

5.5.3. DeVasys USBI2CIO Board Information

A USB interface board is used in this project, which is a Devasys product. The USBI2CIO interface board provides a simple "drop-in" solution for customers that need to connect hardware to a PC [12]. The board provides 20-bit of user-configurable digital I/O, a 90Kbps I^2C interface, and an 8-bit width data bus for fast FIFO transfers. The board uses a Cypress AN2131QC EZUSB microprocessor.

It is a very convenient development environment because it allows loading the designed firmware into the EZUSB microprocessor's RAM through the USB bus. Also, the development board has the option for the bus-powered or self-powered functionality, which makes development fast and easy, and when the device needs to self-sufficient, a jumper is switched and +5V can be supply to the +5V wire loop that is present on the board as a terminal.

The USB enumeration of the Devasys board is a two-steps process. The first step loads a special driver that prepared the device and connection to load the firmware supplied by Devasys. Then, the board forces disconnect and reconnect, called ReNumeration, and then can be detected as USBI2CIO board. Loading the board in this configuration allows the I/O pins on the development board to be controlled through Windows API calls, supported by Visual C++. The Windows API calls allow initializing a board; configure its I/O pins for input or output, and reading and writing the values.

In this project, 12 directly digital I/O pins are used and configured as output. A 12-bit digital voltage output is transferred via this I/O port from host PC to controller

interface board. The I^2C communication interface is used for 8 bytes data array reading from controller interface board to host PC.

All USB communication is controlled by user software in host PC. The code is developed by C Language and divided into two parts. Figure 5.18 shows the flow diagram.

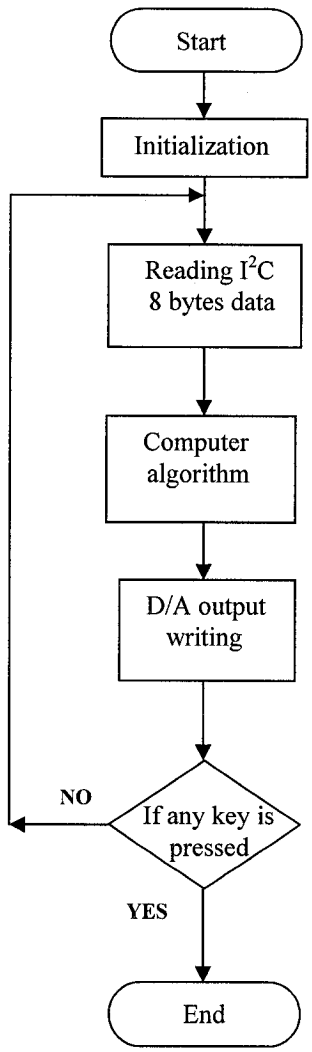

Figure 5.18 Data flow diagram

Its first part is USB initialization for USB configuration setting, and the second part is an unlimited loop. Once the system goes into real time control status, the software will run the loop and continue to exchange data between PC and controller interface board in constant interval time. This data flow will be used by control algorism resided in PC for feedback and output data.

5.6 Controller interface board Circuit Diagram

Figure 5.19 shows the new controller interface board circuit diagram. In the circuit, U1 U2and U3 are octal 3 states non-inverting bus transceiver ICs, which are used in the data bus between microprocessor and USB interface card. There are two purposes to use these interface ICs:

1) To eliminate some of noise and prevent unexpected high voltage spick into microprocessor.
2) Adjust voltage level between microprocessor and USB interface card. Because USB card operates on 3.3V and microprocessor operates on 5V, this causes a mismatch in voltage level at which logical values are read. Sometime when USB is running at 3.15V to microprocessor for the logical high, the transition is missed. Therefore a voltage level adjustment is required.

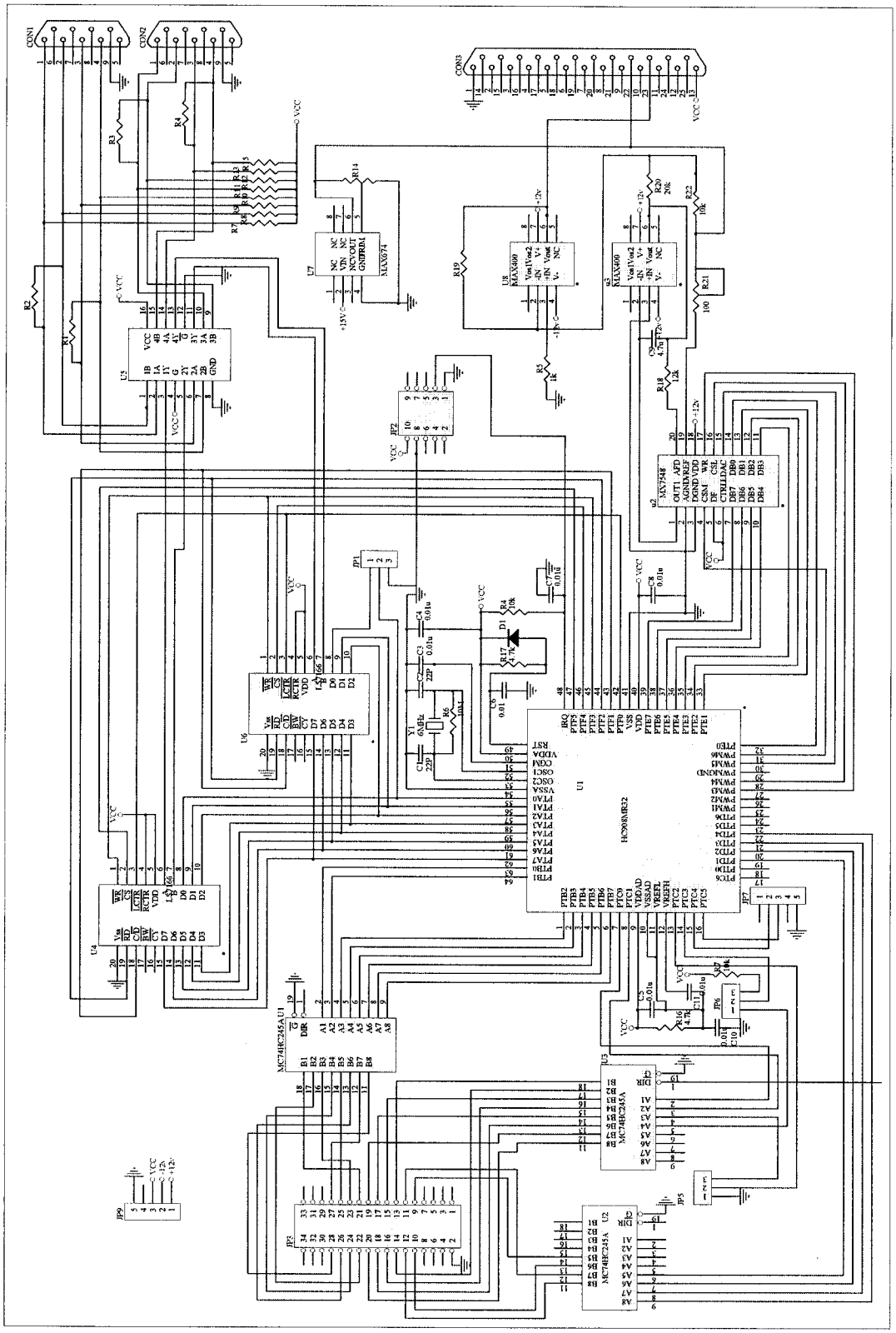

Figure 5.19 The circuit diagram of the new controller interface board

Chapter 6

Unit Test and Verification

6.1. I^2C Interface to Microprocessor

I^2C communication is built between USBI2CIO board and MR32. USBI2CIO board has on-chip hardware I^2C model and acts as a master device. But the MR32 does not have a dedicated hardware for I^2C. In this project, effective software I^2C slave implementation is used in MR32. Therefore the I^2C communication needs to be tested first to determine if it works properly.

The assembly code of I^2C slave application was tested using MR32 emulator system. The reason to use MR32 emulator is to debug assembly code easily. By using the breakpoint feature of the emulator system, any register value and port status are easy to be reviewed after the code executed.

Testing the I^2C communication requires three steps, one is loading the firmware onto the MR32 emulator board, second is connecting MR32 emulator and USBI2CIO board with a 2-twisted wire as I^2C link, third is running the PC host application.

The computer application software was developed for this test as well. An I^2C reading cycle starts by calling the ReadI2C( ) function in this program. The data received by USBI2CIO board via I^2C port will be transmitted to PC through USB interface and can be display on screen immediately. This testing software pseudo code is given as below:

```
While( !any key is pressed)          //if not any key is pressed, continue the
                                     unlimited loop
{
        ReadI2C( &data array); // read an 8 bytes data array from I²C
```

```
time_new=data[6]*256+data[7];      // calculate 16 bits new
                                    time counter value
if(time_new<time_old)
{
    time different=65535+time_new-time_old; // because maximum time
                                              counter value is
                                              65535
}
else
{
        time different=time_new-time_old;
}

        for(i=0 to 8)          //display all 8 bytes data in array
        {
            print data array number
        }
        print time different between old and new time;

}
```

In each I^2C reading cycle, an array with 8 bytes data can be transmitted at a time. The last two bytes of this array are the 16-bit time counter value. The time interval between two I^2C readings can be obtained by means of calculating differential of last and present time counter value. This value is display out on the screen and used to test the speed of whole system communication.

An oscilloscope is used to measure the SCL and SDL signal of I^2C. The waveforms of the signal shown on oscilloscope indicate that the clock signal and data transmitter signal all can be transmitted correctly. A timing diagram is recorded from oscilloscope and given in figure 6.1.

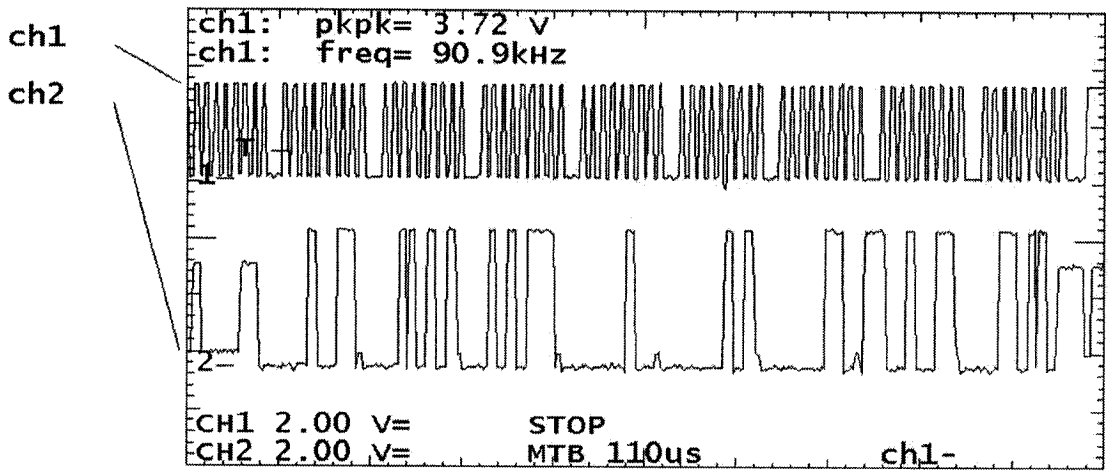

Figure 6.1 I^2C timing diagram

6.2. D/A Converter Test

The purpose to test communication and control between computer and D/A converter is to ensure that MR32 can pass data to D/A converter properly and D/A converter can create correct and accurate analog signal based on digital input.

As discussed before, the computer output is a 12-bit digital data. The output data is transferred to microprocessor via USB interface card first. Microprocessor servers as a peripheral controller and passes the digital output to D/A converter. Because D/A signal transmission relates to different hardware, each individual functional block was tested separately first to determine if it could function properly and then the overall function of D/A converter was tested. The test work is divided into several parts. They are:

1) D/A converter hardware test

Ensure each IC in the D/A converter circuit is powered by 5V and grounded properly. A precision reference voltage generated by MAX674 was tested to determine if

the reference voltage (+10V) is accurate and stable. The reference voltage output of MAX674 was carefully trimmed by tuning the adjustable resistor R14.

2) Test the communication between microprocessor and MX7548.

This is a partial function test for D/A converter. The purpose is to ensure that the interface design between Microprocessor and MX7548 are correct in hardware view and the microprocessor can control and communicate with MX7548 properly.

An assembly code was loaded into microprocessor special for this testing purpose. This code was modified from actual microprocessor control software. Instead of reading D/A data from host computer, microprocessor initials a constant D/A value and writes to D/A converter directly.

The signals on control pin of microprocessor were monitored using an oscilloscope. A writing cycle timing diagram was verified and corrected by studying the control signal waveforms generated on oscilloscope.

The analog voltage output of D/A converter was measured by using a voltage meter. The measured voltage value was compared with calculated value based on the table 5.1. The voltage output value needs to be carefully calibrated and tuned by adjusting the resistor R21. By setting different initial value in microprocessor, several testing points were obtained and tested. The results are listed in table6.1.

Digit input	Analog output	Test output
1111 1111 1111	9.995	9.998
1110 0000 0000	7.5	7.498
1100 0000 0000	5.0	5.001
1010 0000 0000	2.5	2.497
1000 0000 0000	0	0
0110 0000 0000	-2.5	-2.498
0100 0000 0000	-5.0	-4.999
0010 0000 0000	-7.5	-7.500
0000 0000 0000	-10.0	-10.01

Table 6.1 D/A converter analog output test result

3) Full function test of D/A converter

A testing digital data signal varying in SIN wave was created by the computer and transferred to microprocessor through a USB interface. D/A converter received the input data from microprocessor and converted it to an analog signal output. This analog output was recoded by oscilloscopes to see if the voltage output is continuously and varies between +10V to –10 V in SIN wave.

The pseudo code used for this test is given below:

```
//Start here
Configuerate I/O to be output
While(!any key is pressed )    //unlimited loop
{
    if(n==1000)  // divided 360 degree by 1000 steps
        {
            reset n=0;
```

```
        }
        else
        {
                increase n;
        }

    voltage=2047-2047*sin(2*3.1415926*n/1000);
WriteIO( voltage); //output voltage via 12 bit I/O
    }
```

An analog output graphic of D/A converter is given in Figure 6.2.

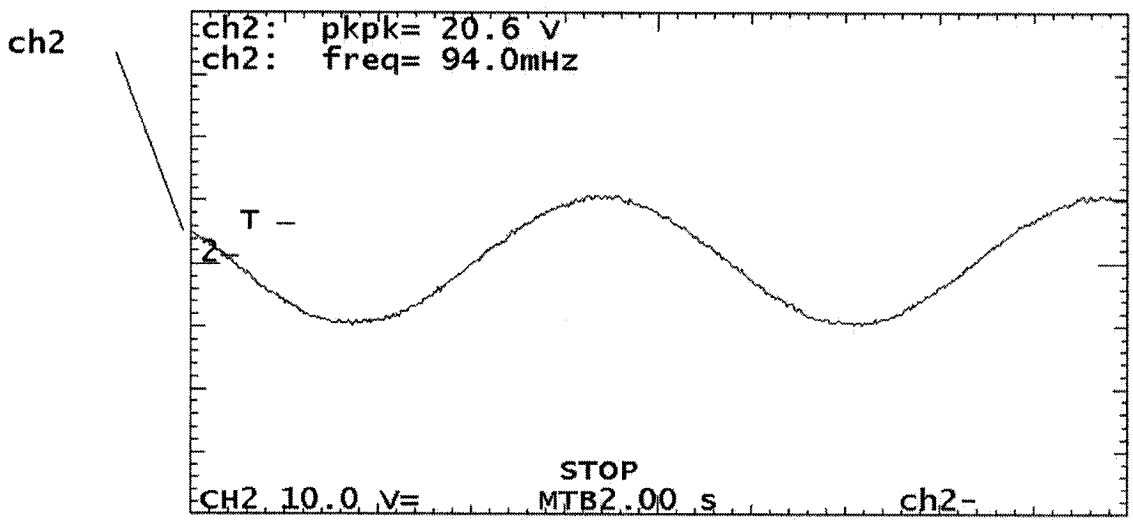

Figure 6.2 An analog output graphic of D/A converter

The figure 6.2 shows that the D/A converter can convert a digital data to an analog signal correctly and the communication between computer and D/A converter works properly as well.

Chapter7

Pendubot System Dynamics Analysis

7.1. Pendubot Model

The equation of motion for the Pendubot can be found using Lagrangian dynamics. This provides a mathematical model that can be used to derive various controllers and to simulate the response. The coordinate description of the Pendubot is given in Figure 7.1

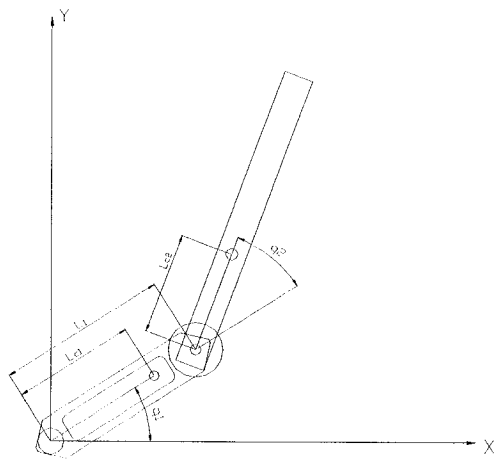

Figure 7.1 Coordinate description of the Pendubot.

Equations of motion for the Pendubot can be simply written in matrix form[4]:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \begin{bmatrix} \tau \\ 0 \end{bmatrix} \tag{7.1}$$

where, $D(q)$ is symmetric and positive define, and

$$D(q) = \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}$$

$$d_{11} = m_1 l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos q_2) + I_1 + I_2 \tag{7.2}$$

$$d_{12} = d_{21} = m_2(l_{c2}^2 + l_1 l_{c2}\cos q_2) + I_2$$

$$d_{22} = m_2 l_{c2}^2 + I_2$$

$C(q,\dot{q})$ has been chosen such that $\dot{D} - 2C$ is skew symmetric.

$$C(q,\dot{q}) = \begin{bmatrix} h\dot{q}_2 & h\dot{q}_2 + h\dot{q}_1 \\ -h\dot{q}_1 & 0 \end{bmatrix}$$

$$h = -m_2 l_1 l_{c2}\sin q_2 \tag{7.3}$$

Note the 0 in the vector on the right side of Eq. (7.1), indicating the absence of an actuator at the first joint. The sign τ is the vector of torque applied to the links and q is the vector of joint angle positions.

Others,

$$g(q) = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}$$

$$\phi_1 = (m_1 l_{c1} + m_2 l_1)g\cos q_1 + m_2 l_{c2} g\cos(q_1 + q_2) \tag{7.4}$$

$$\phi_2 = m_2 g l_{c2}\cos(q_1 + q_2)$$

m_1 : the total mass of link one.

l_1 : the length of link one (See Figure 7.1).

l_{c1} : the distance to the center of mass of link 1 (See Figure 7.1).

I_1 : the moment of inertia of link one about its centroid.

m_2 : the total mass of link two.

l_2 : the length of link two (See Figure 7.1).

l_{c2} : the distance to the center of mass of link 2 (See Figure 7.1).

I_2 : the moment of inertia of link one about its centroid.

g : the acceleration of gravity.

From the above equations it is observed that the seven dynamic parameters can be grouped into the following five parameters equations [2], [3]:

$$\begin{aligned}
\theta_1 &= m_1 l_{c1}^2 + m_2 l_1^2 + I_1 \\
\theta_2 &= m_2 l_{c2}^2 + I_2 \\
\theta_3 &= m_2 l_1 l_{c2} \\
\theta_4 &= m_1 l_{c1} + m_2 l_1 \\
\theta_5 &= m_2 l_{c2}
\end{aligned} \tag{7.5}$$

Substituting these parameters of (7.5) into the Eqs. (7.2)-(7.4) leaves the following matrices

$$D(q) = \begin{bmatrix} \theta_1 + \theta_2 + 2\theta_3 \cos q_2 & \theta_2 + \theta_3 \cos q_2 \\ \theta_2 + \theta_3 \cos q_2 & \theta_2 \end{bmatrix}, \tag{7.6}$$

$$C(q,\dot{q}) = \begin{bmatrix} -\theta_3 \sin(q_2)\dot{q}_2 & -\theta_3 \sin(q_2)\dot{q}_2 - \theta_3 \sin(q_2)\dot{q}_1 \\ \theta_3 \sin(q_2)\dot{q}_1 & 0 \end{bmatrix}, \tag{7.7}$$

$$g(q) = \begin{bmatrix} \theta_4 g \cos q_1 + \theta_5 g \cos(q_1 + q_2) \\ \theta_5 g \cos(q_1 + q_2) \end{bmatrix}. \tag{7.8}$$

Finally, using the invertible property of the mass matrix $D(q)$, the state equations are given by

$$\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = D(q)^{-1}\tau - D(q)^{-1}C(q,\dot{q})\dot{q} - D(q)^{-1}g(q) \tag{7.9}$$

$$x_1 = q_1, \quad x_2 = \dot{q}_1, \quad x_3 = q_2, \quad x_4 = \dot{q}_2$$

$$\dot{x}_1 = x_2 \tag{7.10}$$

$$\ddot{x}_2 = \ddot{q}_1$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \ddot{q}_2$$

7.2. Identification of Pendubot Parameters

For a controller design that neglects friction, these five parameters are all needed. There is no reason to go a step further and find the individual parameters since the control equations can be written with only the five parameters. Mechatronic Systems, Inc. performed on-line identification experiments using the Hamiltonian based energy equation method [1]. The first step in this approach is to write the total energy as the sum of the kinetic energy and potential energy as

$$E = \frac{1}{2}\dot{q}^T D(q)\dot{q} + V(q) \tag{7.11}$$

where $D(q)$ is the inertia matrix that is defined in (7.1) and $V(q)$ represents the potential energy due to the gravity. The total energy E is linear in the inertia parameters and so may be written as

$$E = W(q,\dot{q})\theta \tag{7.12}$$

Where θ is the parameterization defined in (7.5).

Using the well-known passivity or skew-symmetry property, then

$$\dot{E} = \dot{q}^T \tau \tag{7.13}$$

Between any two times $t = T$ and $t = T + dT$, the equation (7.13) will be:

$$dE = E(T + dT) - E(T) = \int_{T}^{T=dT} \dot{q}^T(u)\tau(u)du = \{W(T + dT) - W(T)\}\theta \tag{7.14}$$

Eq. (7.14) can be used with a standard least squares algorithm to identify the parameter vector θ by applying an open loop signal $t \to \tau(t)$ and recording $\tau, q, \dot{q}$ over a given time interval. Carrying out this identification procedure using a step input to excite the Pendubot resulted in the following parameter values,

$$\theta_1 = 0.0308 Volts.\sec^2$$

$$\theta_2 = 0.0106 Volts.\sec^2$$

$$\theta_3 = 0.0095 Volts.\sec^2$$

$$\theta_4 = 0.2087 Volts.\sec^2/m$$

$$\theta_5 = 0.0630 Volts.\sec^2/m$$

7.3. The Pendubot Classical Control Algorithm

A classical control algorithm was run on the new designed controller for Pendubot swing up control. This algorithm is provides by Mechatronic Systems, Inc. [1]. Because

this algorithm has be proved on previous controller that it is a successful algorithm and can swing and balance the links at equilibrium position easily, it would be the best algorithm to be used to test the new controller.

To simplify the problem, this control is divided into two parts, swing up control, and balancing control. First the controller uses the swing control algorithm to swing the links from their stable hanging position to unstable equilibrium and then switch to balance control algorithm to balance links at equilibrium. The swing up control uses the partial feedback linearization method. And balance control uses a method that is similar to the classical cart-pole inverted pendulum.

7.4. Swing up Control

The equations of motion of the Pendubot are given by Eq. (7.1). Neglecting the friction constants and performing the matrix and vector multiplications, the equations of motion are rewritten as:

$$d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + c_{11}\dot{q}_1 + c_{12}\dot{q}_2 + \phi_1 = \tau_1 \tag{7.15}$$

$$d_{21}\ddot{q}_1 + d_{22}\ddot{q}_2 + c_{21}\dot{q}_1 + \phi_2 = 0 \tag{7.16}$$

In order to linearize one of the degrees of freedom, an outer loop control that tracks a given trajectory for the linearized degree of freedom is used. In the case of the Pendubot, the collocated degree of freedom q_1 was chosen to linearize. Eq. (7.16) was rewritten for link two:

$$\ddot{q}_2 = (-d_{21}\ddot{q}_1 - c_{21}\dot{q}_1 - \phi_2)/d_{22} \tag{7.17}$$

Substituting Eq. (7.17) into Eq. (7.15) yields

$$\bar{d}_{11}\ddot{q}_1 + \bar{c}_{11}\dot{q}_1 + \bar{c}_{12}\dot{q}_2 + \bar{\phi}_1 = \tau_1 \tag{7.18}$$

with

$$\bar{d}_{11} = d_{11} - \frac{d_{12}d_{21}}{d_{22}}$$

$$\bar{c}_{11} = c_{11} - \frac{d_{12}c_{21}}{d_{22}} \tag{7.19}$$

$$\bar{c}_{12} = c_{12}$$

$$\bar{\phi}_1 = \phi_1 - \frac{d_{12}\phi_2}{d_{22}}$$

By using full linearization method (also called the computed torque method [24]), the inner loop control that linearizes q_1 can be defined as

$$\tau_1 = \bar{d}_{11}v_1 + \bar{c}_{11}\dot{q}_1 + \bar{c}_{12}\dot{q}_2 + \bar{\phi}_1 \tag{7.20}$$

This results in the system

$$\ddot{q}_1 = v_1 \tag{7.21}$$

$$d_{22}\ddot{q}_2 + c_{21}\dot{q}_1 + \phi_2 = -d_{21}v_1 \tag{7.22}$$

Since the equation is now linear, an outer loop control can be designed to track a given trajectory for link one. The response of link two then is given by the resulting nonlinear equation (7.22). This equation represents internal dynamics with respect to an output $y = q_1$. The goal of the outer loop control is to track a given trajectory for link one

and at the same time excite the internal dynamics to swing link two to a balance position. A PD with feedforward acceleration was chosen by MSI for application in the Pendubot.

$$v_1 = \ddot{q}_1^d + K_d(\dot{q}_1^d - \dot{q}_1) + K_p(q_1^d - q_1) \tag{7.23}$$

See Figure 7.2 for a block diagram of the swing up control.

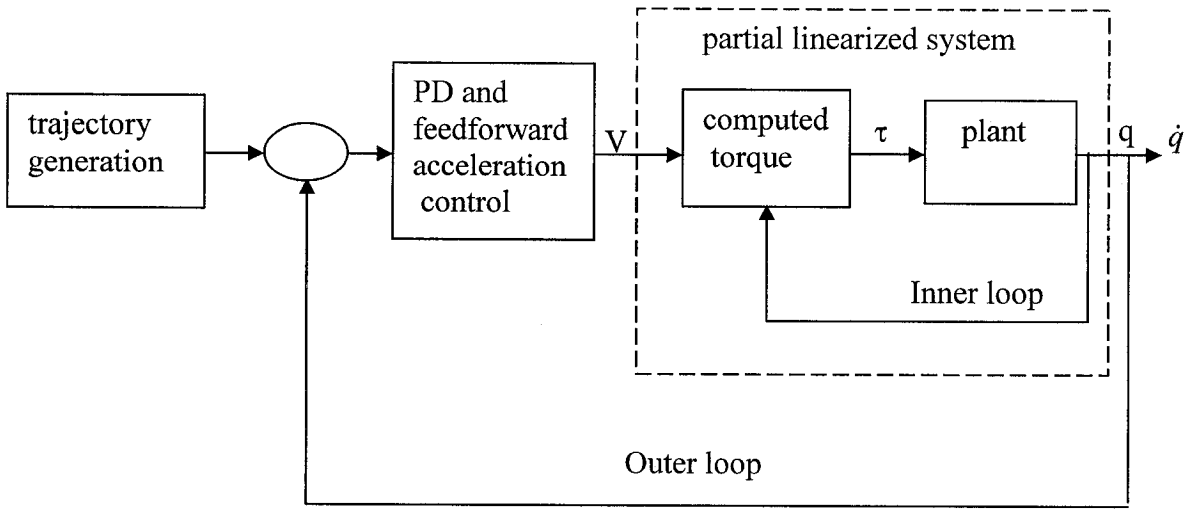

Figure 7.2 Block diagram of the Partial Feedback Linearization Control

With this controller, a trajectory must be determined to swing the links to their unstable equilibrium position. To swing the links to the upright top position ($q_1 = \pi/2$, $q_2 = 0$), step trajectory $q_1 = \pi/2$ is used by MSI. MSI also gives a trajectory for the mid position

$$q_1 = 1.4 \sin(5t) - \pi/2 \quad t < 2\pi/5$$

$$q_1 = -\pi/2 \quad t > 2\pi/5$$

This trajectory pumps energy into the system by causing the second link to swing back and forth and finally up to its middle equilibrium point.

The K_p and the K_d gains need to be adjusted carefully in late test in order to swing the second link slowly into its equilibrium position so that another controller can catch the link.

7.5. Balancing Control

The pendubot's nonlinear equations of motion (7.16) was linearized by using the Taylor series approximation, the equation can be rewritten as

$$f_a(x,u) = f_a(x_r,u_r) + \frac{\partial f}{dx}|_{x_r,u_r}\ (x - x_r) + \frac{\partial f}{dx}|_{x_r,u_r}\ (u - u_r) \qquad (7.24)$$

Where:

x is the vector of states.

U is the single control input for the Pendubot

x_r and u_r are the equilibrium values of the states and control.

Since the interest is only in controlling the Pendubot at equilibrium positions, f_a (x_r, u_r) will always be zero. All that is needed then is to find out the partial derivative matrices and evaluate them at the equilibrium points. The Eqs.(7.4) - (7.8) shows that the Pendubot's equilibrium points can be defined by:

$$u_r = \theta_4 g \cos(x_{r1}) \qquad (7.25)$$

$$x_{r1} + x_{r3} = \pi / 2 \qquad (7.26)$$

Differentiating equation with respect to the states leaves the A matrix in the linearized model:

$$\frac{df}{dx} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{df_2}{dx_1} & 0 & \frac{df_2}{dx_3} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{df_4}{dx_1} & 0 & \frac{df_4}{dx_3} & 0 \end{bmatrix} \tag{7.26}$$

The B matrix is found by the partial derivative with respect to the control input:

$$\frac{df}{du} = \begin{bmatrix} 0 \\ \frac{df_2}{du} \\ 0 \\ \frac{df_4}{du} \end{bmatrix} \tag{7.27}$$

Each equilibrium point defines a different linearized system. This means that different control gains will be needed for each equilibrium point for best results in balancing of the Pendubot. The top balancing position as the upright position was defined as $x_{r1} = \pi/2$, $x_{r3} = 0$ and $u_r = 0$. The middle balancing position is define as $x_{r1} = -\pi/2$, $x_{r3} = \pi$ and $u_r = 0$.

Using these equilibrium values and the parameters identified by the energy equation method, the linear models for the top and mid equilibrium position are as following:

Top

$$\dot{X} = AX + Bu$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 51.9265 & 0 & -13.9704 & 0 \\ 0 & 0 & 0 & 1 \\ -52.8402 & 0 & 68.4210 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 15.9549 \\ 0 \\ -29.3596 \end{bmatrix}$$

Mid

$$\dot{X} = AX + Bu$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -51.9265 & 0 & 13.9704 & 0 \\ 0 & 0 & 0 & 1 \\ 51.0128 & 0 & 40.4801 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 15.9549 \\ 0 \\ -2.5502 \end{bmatrix}$$

With these linear models the LQR or pole placement techniques can be used to design full state feedback controllers.

7.6. Software

The control algorithms were written in C code. Its source code is given in Appendix II. Figure7.3 shows the flow chart of the control algorithm.

First the controller runs the swing up algorithm and waits for link one to arrive within 0.1 radians of its equilibrium position and then checks link two. If link two is also within 0.20 radians of its equilibrium position, then the control algorithm switch to the balancing control. Some protection features are added in program. If the control output is less than 9volts, 10 volts being the maximum DAC output for the Pendubot, the control is switched to the balancing mode. Otherwise the links are passing too quickly though the equilibrium point and the swing up control remains intact.

The velocity of join is obtained by using the finite difference method, it is

[x(k)-x(k-1)]/sample period

This creates numerical error or noise in the calculation of the control effort due to the finite resolution of the optical encoders. In order to reduce the error, an average velocity calculation is taken from last three samples. The velocity calculation for joint one is written as follows:

A negative voltage U=-1V is added into the control in first time swing up. This sends the motion of link in the negative direction for a shot of time to add potential energy into system for swing up. The reason to do this is to overcome the motor torque when the system is powered up.

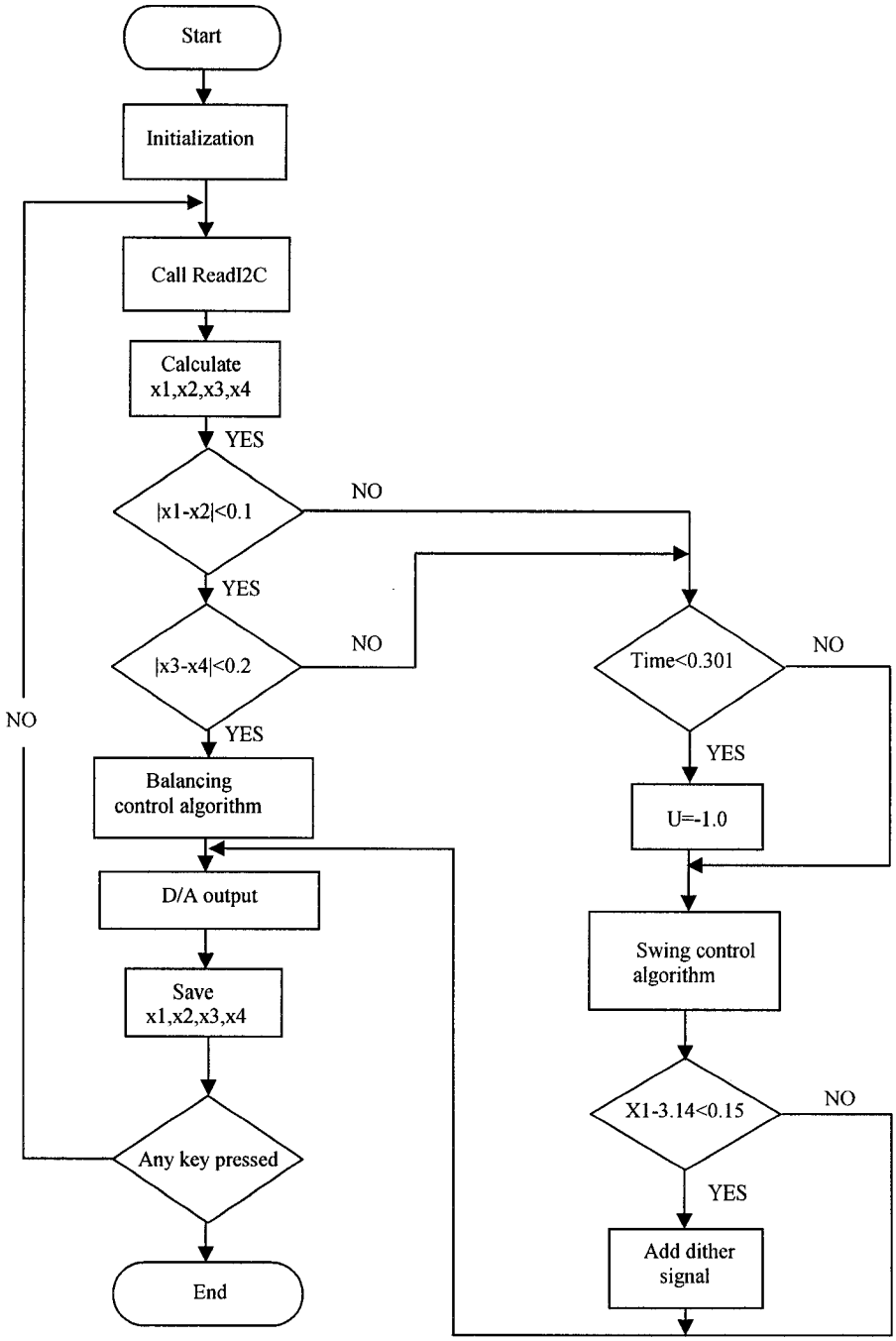

Figure 7.3 The flow chart of the control algorithm.

A dither signal was also needed to help balance the links in the top position. Due to friction and the increased effect of gravity on the links in the top position, the balancing control was not able to hold the links motionless. The balance control at the mid position did not have this problem. The main reason for this is that gravity works with the control at the mid position to keep link one at equilibrium. At the top position, gravity works to pull both of the links away from the equilibrium. The motion produced was large amplitude away (approximately 0.2 radians). To eliminate some of this motion, an open-loop dither signal was added to the control.

$$U = -K_x + 0.25\sin 40.0t.$$

This dither signal helped to eliminate much of the sway but it was not able to cancel all of the motion.

7.7. Experimental Result

The section shows actual response of the Pendubot system by using above control algorithm with new control system. The Figure 7.4 shows the Pendubot swinging up and balancing at the top position. The Figure 7.5 shows the Pendubot swinging up and balancing at middle position.

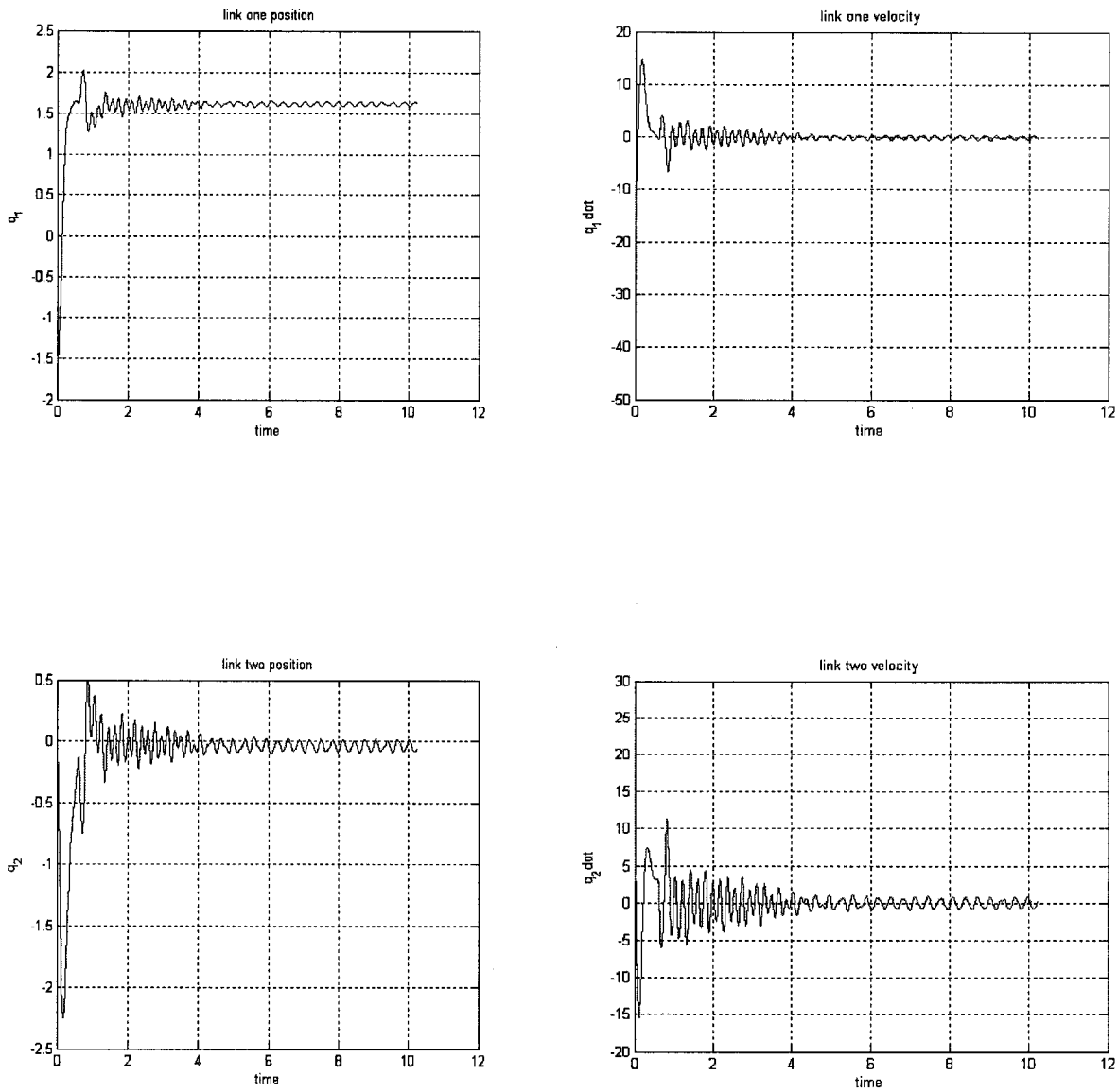

Figure 7.4 Experiment results of balancing Pendubot at top

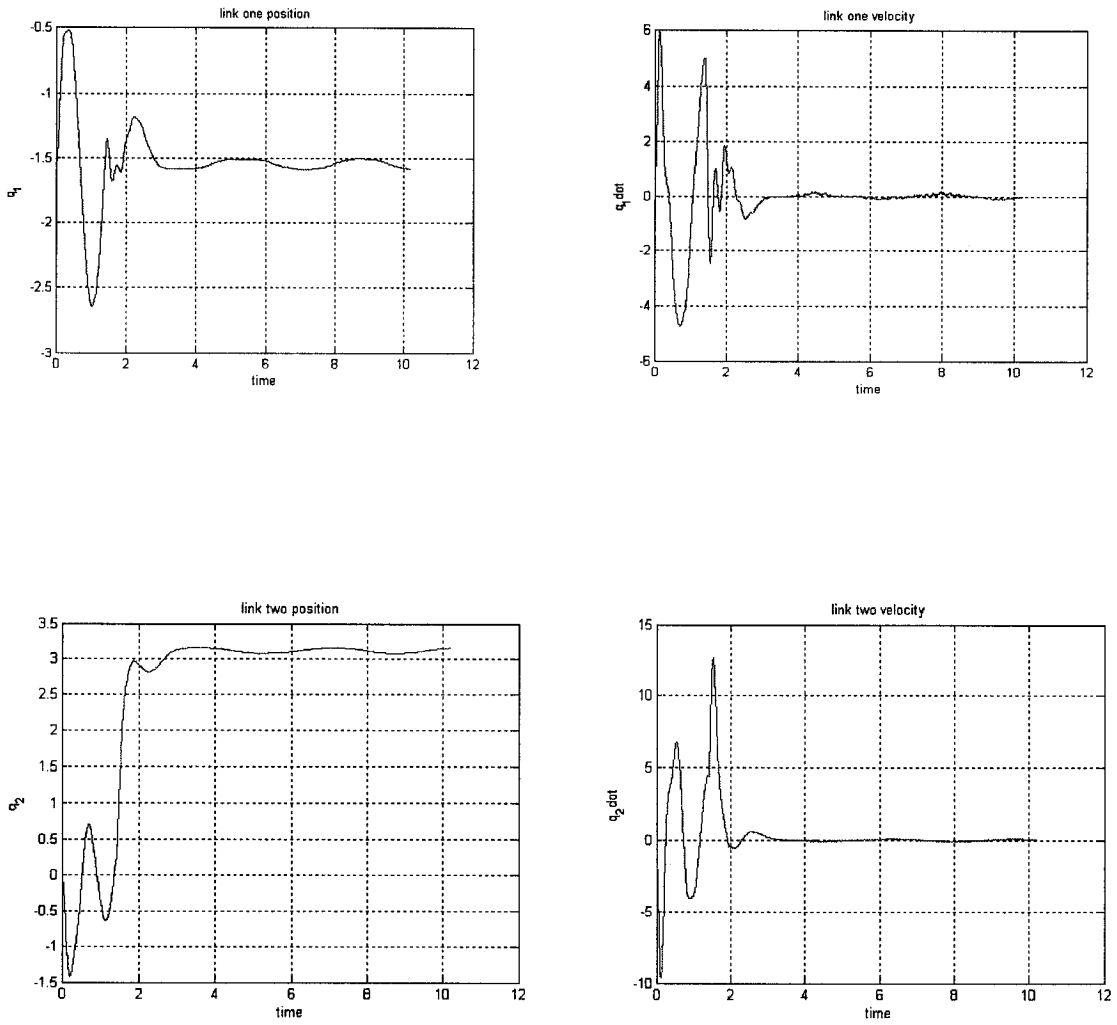

Figure 7.5 Experiment results of balancing Pendubot at middle position

Chapter 8

Conclusion and Future Work

This thesis is focused on the electronic design of a Pendubot controller interface board based on previous MSI Pendubot experimental system. A new Pendubot controller interface board has been proposed and designed. A USB interface is used to serves as the device interface to communicate with PC. The board consists of a D/A converter and two encoder counters, which are controlled by on-board microprocessor.

A detail circuit design of the board in accord with user software and hardware firmware is provided in this thesis. A prototype of board is made and its performance was tested and verified.

A classical Pendubot control algorithm for balancing the link at top and mid position was running on the system.

The experiment results show that the control device with proper control algorithm resided in PC can control the Pendubot very well.

In this thesis, all the user application software is written by C code with simple DOS interface. In future work, a window interface will be better for user to operate and modify a controller resided in PC easily. An internal link to the application software of MATLAB should be added in order to plot out testing result directly in future work.

So far, only a classical control algorithm was conducted on the control system. In next step, more control algorithms should be implemented and tested on the new control system.

Reference

[1] D. J. Block, "Mechanical Design and Control of the Pendubot", M.S.Thesis, Department of General Engineering, University of Illionis at Urbana-champain, 1996

[2] M. W. Spong "The Swing Up Control Problem for The Acrobot", *IEEE Transactions on Control Systems Magazine*, pp.49-55, February 1995..

[3] N.S. Dedrossian, "Nonlinear Control of an Under-actuated Two Link Manipulator", *Proceedings of the 32nd Conference on Decision and Control,* pp.2234-2238, 1993.

[4] A.Isidori, "Nonlinear Control Systems", *3rd ed. Springer-Verlag*, 1995

[5] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*, pp. 145-148, 1989.

[6] M. W. Spong, "The Control of Underactuated Mechanical Systems", *Plenary Lecture at the first International conference on Mechatronics*, Mexico City, January 26-29, 1994

[7] A. Isidori, C. I. Bynes and F. Flockerzi "Topics in Control Theory", *Berkhauser* 1994.

[8] "*128K I2C™CMOS Serial EEPROM.* (24LC128)", Microchip Technology Inc. 1998

[9] "AM26LS32AC, AM26LS32AI, AM26LS33AC, AM26LS32AM, AM26LS33AM, Quadruple Differential Line Receivers". Texas Instruments Incorporated, 2002

[10] G. Whitacre "Using MC68HC908 On-Chip FLASH Programming Routines" Motorola, Inc. 2001

[11]"EZ-USB Technical Reference Manual", Cypress Semiconductor Corporation. 2002

[12] "API User's Guide for the USB I^2C/IO Universal Serial Bus Interface Module", DeVasys, 2001,2002

[13] "BX30A Series Brushless Servo Amplifiers Model BX25A20", Advanced Motion Controls Inc.

[14] "CIO-DAC02 DUAL Channel 12 Bit Analog Output USER'S MANUAL". Measurement Computing Corporation.

[15] "Software I2C™Slave Implementation". ATMEL

[16] "Addendum to the M68ICS08 Operator's Manual M68ICS08SOM/D for ICS08JB", P&E Microcomputer Systems, Inc, 2000.

[17] "LS7166 Encoder to Microprocessor Interface Chip". US Digital Corporation,2001.

[18] "68H908MR32 General Release Specification", Motorola, APRIL 2,1998

[19] "MC74HC245A Octal 3-State Noninverting Bus Transcelver". Semiconductor Components Industries, 2000.

[20] S. Arendarik, "I2C Slave on the HC908QT/QY Family MCU", Motorola, Inc. 2003.

[21] "CMOS 8-Bit μP Compatible 12-Bit DAC FOR MX7548",Maxim Integrated Products.

[22] B. C. Kuo, "Computer Software for Control Systems ACSP Version 2.0" ISBN 0-13-304759-8

[23] "Universal Serial Bus Specification", Revision 1.1,September 23,1998

Appendix A

```
     ORG $8000
;GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
;          MR32 control and I2C communication code
;                              Li Zhang
;                              version 3.0
;                              Data March 17, 2004
;GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
PORTA  EQU 0000H
PORTB  EQU 0001H
PORTC  EQU 0002H
PORTD  EQU 0003H
DDRA   EQU 04H
DDRB   EQU 05H
DDRC   EQU 06H
          ;07H IS UNIMPLEMENTED
PORTE  EQU 08H
PORTF  EQU 09H
          ;0A,0B ARE UNIMPLEMENTED
DDRE   EQU 0CH
DDRF   EQU 0DH

TASC   EQU 0EH
TACNTH EQU 0FH
TACNTL EQU 10H
TAMODH EQU 11H
TAMODL EQU 12H
TASC0  EQU 13H
TACH0H EQU 14H
TACH0L EQU 15H
TASC1  EQU 16H
TACH1H EQU 17H
TACH1L EQU 18H

TASC2  EQU 19H
TACH2H EQU 1AH
TACH2L EQU 1BH
TASC3  EQU 1CH
TACH3H EQU 1DH
TACH3L EQU 1EH

CONFIG EQU 1FH
PCTL1  EQU 20H
PCTL2  EQU 21H
FCR    EQU 22H
FSR    EQU 23H
FTACK  EQU 24H

PWMOUT EQU 25H
PCNTH  EQU 26H
```

```
PCNTL  EQU 27H
PMODH  EQU 28H
PMODL  EQU 29H
PVAL1H EQU 2AH
PVAL1L EQU 2BH
PVAL2H EQU 2CH
PVAL2L EQU 2DH
PVAL3H EQU 2EH
PVAL3L EQU 2FH
PVAL4H EQU 30H
PVAL4L EQU 31H
PVAL5H EQU 32H
PVAL5L EQU 33H
PVAL6H EQU 34H
PVAL6L EQU 35H

DEADTM EQU 36H
DISMAP EQU 37H
SCC1   EQU 38H
SCC2   EQU 39H
SCC3   EQU 3AH
SCS1   EQU 3BH
SCS2   EQU 3CH
SCDR   EQU 3DH
SCBR   EQU 3EH

ISCR   EQU 3FH
ADSCR  EQU 40H
ADRH   EQU 41H
ADRL   EQU 42H          ;ADR IS 8 BIT IN MP16
ADCLK  EQU 43H
SPCR   EQU 44H
SPSCR  EQU 45H
SPDR   EQU 46H
                        ;47H-50H UNIMPLEMENTED

TBSC   EQU 51H
TBCNTH EQU 52H
TBCNTL EQU 53H
TBMODH EQU 54H
TBMODL EQU 55H
TBSC0  EQU 56H
TBCH0H EQU 57H
TBCH0L EQU 58H
TBSC1  EQU 59H
TBCH1H EQU 5AH
TBCH1L EQU 5BH

PCTL   EQU 5CH
PBWC   EQU 5DH
PPG    EQU 5EH
                    ;5F UNIMPLEMENTED
SBSR   EQU $FE00
SRSR   EQU $FE01
SBFCR  EQU $FE03
FLCR   EQU $FE08
```

```
BRKH   EQU $FE0C
BRKL   EQU $FE0D
BRKSCR EQU $FE0E
LVISR  EQU $FE0F
FLBPR  EQU $FE7E
COPCTL EQU $FFFF



;GGGGGGGGGGGGGGGGGGGGGGGGGGGGGG

;    RAM LABELS
DEV_ADDR EQU 61H        ;INTERNAL DEVICE ADDRESS
WR_BYTE1 EQU 62H        ;HIGH ADDRESS BYTE FOR DATA TO BE WRITTEN
WR_BYTE2 EQU 63H        ;LOW ADDRESS BYTE FOR DATA TO BE WRITTEN
WR_BYTE3 EQU 64H        ;DATA TO BE WRITTEN
RD_BYTE  EQU 65H        ;DATA FROM CURRENT ADDRESS
REG1     EQU 66H        ;AUXILIARY REGISTER
NUM_BIT  EQU 67H        ;NUMBER OF RECEIVED BITS
COUNT1   EQU 68H        ;COUNTER REGISTER 1
COUNT2   EQU 69H        ;COUNTER REGISTER 2
R_DATA   EQU 6AH        ;RECEIVE DATA
TEMP     EQU 6BH        ;TEMP BYTE1
ETEMP    EQU 6CH        ;TEMP BYTE2

R_W      EQU 6DH        ;R/W FLAG ON I2C(BIT 0)

INI_FLAG   EQU 6CH      ;initial flag
TEMPO1     EQU 6DH
DAC_H  EQU 6EH          ;DAC OUTPUT HIGH BYTE
DAC_L  EQU 6FH          ;DAC OUTPUT LOW BYTE

;TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT


SCANCNT   EQU 70H

;################
; RAM EXCLUSIVELY FOR TOP ELEMENT CONTROL

RLLEVEL   EQU 71H   ;LEVEL SETTING FOR REAR LEFT ELEMENT   0=OFF
RRLEVEL   EQU 72H   ;LEVEL SETTING FOR REAR RIGHT ELEMENT  0=OFF
FLLEVEL   EQU 73H   ;LEVEL SETTING FOR FRONT LEFT ELEMENT  0=OFF
FRLEVEL   EQU 74H   ;LEVEL SETTING FOR FRONT RIGHT ELEMENT 0=OFF
CENLEVEL  EQU 75H    ;level setting for center element 0=off

STATE     EQU 76H   ;0=RL ON,|1=RR ON|2=FL ON|3=FR ON|4=DUAL ON|
;              ;5=KEY LOCK ON|6= CENTER ON|7= energized   ;+b
DIG       EQU 77H   ;EACH BIT REPRESENTS DIGIT TO BE DISPLAYED NEXT


; DIG1 TO DIG8 BIT FOR BIT TRANSFER TO THE SEGMENT ADDRESSES
; AND ARE USED ONLY FOR THE DISPLAY.
```

```
DIG1    EQU 78H   ;DISPLAY BUFFER FOR DIGIT1
DIG2    EQU 79H   ;DISPLAY BUFFER FOR DIGIT2
DIG3    EQU 7AH   ;DISPLAY BUFFER FOR DIGIT3
DIG4    EQU 7BH   ;DISPLAY BUFFER FOR DIGIT4
DIG5    EQU 7CH   ;DISPLAY BUFFER FOR DIGIT5
DIG6    EQU 7DH   ;DISPLAY BUFFER FOR DIGIT6
DIG8    EQU 7EH   ;display buffer for digit8
DIG1BUF  EQU 7FH
DIG2BUF  EQU 80H
DIG3BUF  EQU 81H
DIG4BUF  EQU 82H
DIG5BUF  EQU 83H
DIG6BUF  EQU 84H   ;BUFFER FOR DIGIT 5 SO NO FLICKERING OCCURS DURING
            ; MANIPULATION
DIG8BUF  EQU 85H

BEEPTOG  EQU 86H   ;BIT 0 SET = BEEP | BIT 1 SET = DO NOT DISPLAY
            ; BIT2 SET = DISPLAY DELAYED DUE TO BIT 1
STATUS   EQU 87H   ;BIT:0=INIT MODE |1 KEYBOARD ERROR |2 CURRENT TEST STARTED
            ;3 ILLIGAL KEY TOUCH |4 wait mode |5 int in progress
            ;6 temp to high flash 1/10|7= MULTIPLE KEYS

KEYS    EQU 88H   ;for 2nd scanning
SLO     EQU 89H   ;PASS COUNTER USED IN "LINES"
DUNLIN   EQU 8AH   ;BIT01=DIG1 DUN "LINES"|1=DIG2 DUN "LINES"|2=DIG3 DUN "LINES"|
            ;3=DIG4 DUN "LINES"|4=DIG5 DUN "LINES"
BPCNT    EQU 8BH   ; BEEPER COUNTER
INTCOUNT  EQU 8CH
HALFTOG  EQU 8DH   ;BIT0 IS TOGGLED EVERY 1/2 SEC
COUNTINT  EQU 8EH   ;INTERUPT COUNTER
COUNTLOCK EQU 8FH   ;10 = 5 SECONDS
MAESTRO  EQU 90H   ;BIT 0=CALL FOR 2 BEEPS |1= high temperature beeps
BEEP2    EQU 91H   ;COUNTER FOR 2BEEPS
RELEASE  EQU 92H   ;bit0=1:key is sure
PROGRAM  EQU 93H   ;BIT: 0=RL|1=RR|2=FL|3=FR|6=CEN  set for starting
SCROLL   EQU 94H   ;BIT: 0=RLUP|1=RLDWN|2=RRUP|3=RRDWN|4=FLUP|5=FLDWN
            ;   6=FRUP|7=FRDWN]
SCROCEN  EQU 95H   ;bit: 0=cenup|1=cendwn
SCRO     EQU 96H   ;SCROLL COUNTER
AK      EQU 97H
PROGOUT  EQU 98H   ;clear when key is pressed
send    equ 99h
;LEVCOUNT  EQU 9AH

AMBT     EQU 9BH
AMBDB    equ 9Ch
beeptry  equ 9Dh ;+b7
keybuf   equ 9Eh ;+b7
nkeycnt  equ 9Fh ;+d
TSCANCNT  EQU $A0
TSCANFLG  EQU $A1
ALASTOP  EQU $A2  ;0 TEMPERATURE HIGH  |1 KEYBOARD ERROR |2 INITIAL ALARM
TIMER
;CHSWCNT  EQU $A3
ADCH     EQU $A4
```

```
KEYERR   EQU $A5
RLTIME   EQU $A6
RRTIME   EQU $A7
FLTIME   EQU $A8
FRTIME   EQU $A9
CNTIME   EQU $AA
PORTA_TMP EQU $AB
PORTB_TMP EQU $AC
PORTC_TMP EQU $AD
PORTE_TMP EQU $AE
PORTF_TMP EQU $AF
DRA_TMP  EQU $B0
DRB_TMP  EQU $B1
DRC_TMP  EQU $B2
DRE_TMP  EQU $B3
DRF_TMP  EQU $B4
COUNTPULSE EQU $B5
ERRSTAT  EQU $B6  ;BIT: 0=RL|1=RR|2=FL|3=FR|6=CEN|7=LOCK & DUAL
SELECT   EQU $B7   ;BIT0:SET=DUAL IN RL, CLR=DUAL IN FL
LINCNT   EQU $B8
CNCOUNT  EQU $B9
FRCOUNT  EQU $BA
FLCOUNT  EQU $BB
RRCOUNT  EQU $BC
RLCOUNT  EQU $BD
PWLOW    EQU $BE  ;0=POWER POSSIBLE LOW, WAIT 1.5 SECONDS TO HAVE CORRECT
NKEY LEVEL
PWLOWCNT EQU $BF
SAFECLR  EQU $D1  ;GG
LOCKCNT  EQU $D2      ;II
SHUTCNT  EQU $D3      ;MM
debou    equ $D4
deboucount equ $D5
;######################
read_byte  EQU $100  ;100-105H IS RESERVED FOR ENCODER READING DATA***
time_h     equ $106  ;106H,107H ARE TIMER COUNTER
time_l     equ $107
write_byte EQU $114  ;114-126H IS RESERVED ;***VREAD
NKEYDEB  EQU $13a  ;13a - 14cH IS RESERVED  ;***

STACK1      EQU $035E


;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$4
RESET:

        sei
        LDHX #STACK1+1
        TXS

        LDA #$66
        STA PPG
        LDA #$3f
        STA PCTL
        LDA #$80
        STA PBWC
```

```
PLL_STABLE:


        BCLR 0,ISCR     ;IRQ FALLING ENDGE ONLY
        MOV #$1d,CONFIG ;DISABLE COP,LVI RESET &ENABLE PWM INDEPENDENT MODE
        CLR DISMAP
        BCLR 0,PCTL1    ;DISABLE PWM
        MOV #$7F,PWMOUT ;CONFIG PWM AS GENERAL PURPOSE OUTPUT, SET ALL PINS
ARE HIGH
        ;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        LDA #0F0H
        STA DDRC
        CLR PORTC
        LDA #00H
        STA DDRB
        CLR PORTB

        ;LDA #0FDH
        ;STA DDRC
        ;CLR PORTC
        LDA #$FF
        STA DDRE
        CLR PORTE

        CLRX
        CLRH
TAG1:
        AIX #1
        CLR $5F,X
        CPHX #$2ff
        BNE TAG1  ;CLEAN 60H-35FH
        CLRX
        CLRH
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

       ;BSET 0,STATUS
SKIP:

        LDA #$01
        CLRX
        STA READ_BYTE,X
        LDA #$02
        INCX
        STA READ_BYTE,X
        LDA #$03
        INCX
        STA READ_BYTE,X
        LDA #$04
        INCX
        STA READ_BYTE,X
        LDA #$05
        INCX
        STA READ_BYTE,X
        LDA #$06
        INCX
        STA READ_BYTE,X
```

```
;************************encoder initial**************************
;       encoder control pins
;           PTF0------LCTR
;           PTF1------C/D
;           PTF2------RD
;           PTF3------WR
;           PTF4------CS2
;           PTF5------CS1
;       commend list
;           MASTER_RESET   0x20
;           INPUT_SETUP    0x68
;           ADDR_RESET     0x01
;           LATCH_CNTR     0x02
;           CNTR_RESET     0x04
;           PRESET_CTR     0x08
;           Error          0xFF
;           QUAD_X         0xC1
;*****************************************************************
;************** SELECT CS1*************
            LDA #$FF
            STA DDRF      ;SET PORTF OUTPUT
            LDA #$1F      ;SELECT CS1 LOW,OTHER HIGH
            STA PORTF     ;ENCODER CS1 ON
            LDA #$FF
            STA DDRA      ;SET PORTA OUTPUT
        ;OUTPUT COMMOND --MASTER RESET
            LDA #$20      ;LOAD COMMEND MASTER_RESET
            STA PORTA
            BCLR 3,PORTF  ;WRITE LOW
            STA PORTA     ;RELOAD PORTA FOR DELAY
            BSET 3,PORTF  ;WRITE HIGH
        ;OUTPUT COMMOND --INPUT RESET
            LDA #$68      ;LOAD COMMEND INPUT_RESET
            STA PORTA
            BCLR 3,PORTF  ;WRITE LOW
            STA PORTA     ;RELOAD PORTA FOR DELAY
        ;OUTPUT COMMOND --QUAD X
            LDA #$C3      ;LOAD COMMEND QUAD 4X
            STA PORTA
            BCLR 3,PORTF  ;WRITE LOW
            STA PORTA     ;RELOAD PORTA FOR DELAY
            BSET 3,PORTF  ;WRITE HIGH    BSET 3,PORTF  ;WRITE HIGH
        ;OUTPUT COMMOND --CNTR RESET
            LDA #$04      ;LOAD COMMEND CNTR_RESET
            STA PORTA
            BCLR 3,PORTF  ;WRITE LOW
            STA PORTA     ;RELOAD PORTA FOR DELAY
            BSET 3,PORTF  ;WRITE HIGH
        ;OUTPUT COMMOND --ADDR RESET
            LDA #$01      ;LOAD COMMEND ADDR_RESET
            STA PORTA
            BCLR 3,PORTF  ;WRITE LOW
            STA PORTA     ;RELOAD PORTA FOR DELAY
            BSET 3,PORTF  ;WRITE HIGH
    ;RELOAD VALUE=30000 OR #7530 IN HEX
         ;OUTPUT DATA1
```

```
        LDA #$30      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
        BSET 1,PORTF  ;C/D HIGH
      ;OUTPUT DATA2
        LDA #$75      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
        BSET 1,PORTF  ;C/D HIGH
      ;OUTPUT DATA3
        LDA #$00      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
        BSET 1,PORTF  ;C/D HIGH
      ;OUTPUT COMMOND --PRESET RESET
        LDA #$08      ;LOAD COMMEND PRESET_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
      ;OUTPUT COMMOND --LATCH COUNT
        LDA #$02      ;LOAD COMMEND LATCH COUNT
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
      ;OUTPUT COMMOND --ADDR RESET
        LDA #$01      ;LOAD COMMEND ADDR_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
; ************** ENCODER1 OFF*************
        LDA #$FF
        STA DDRF      ;SET PORTF OUTPUT
        LDA #$3F      ;SELECT CS1 HIGH,OTHER HIGH
        STA PORTF     ;ENCODER CS1 OFF
        lda #$01
        sta INI_FLAG  ;set initial flag 0
        bset 1,ISCR   ;disable external interrupt

;************ ENCODER 1 INITIALIZATION END*********


;************** SELECT CS2*************
        LDA #$FF
        STA DDRF      ;SET PORTF OUTPUT
```

```
        LDA #$2F      ;SELECT CS2 LOW,OTHER HIGH
        STA PORTF     ;ENCODER CS2 ON
        LDA #$FF
        STA DDRA      ;SET PORTA OUTPUT
    ;OUTPUT COMMOND --MASTER RESET
        LDA #$20      ;LOAD COMMEND MASTER_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
    ;OUTPUT COMMOND --INPUT RESET
        LDA #$68      ;LOAD COMMEND INPUT_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
    ;OUTPUT COMMOND --QUAD X
        LDA #$C3      ;LOAD COMMEND QUAD 4X
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH    BSET 3,PORTF  ;WRITE HIGH
    ;OUTPUT COMMOND --CNTR RESET
        LDA #$04      ;LOAD COMMEND CNTR_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
    ;OUTPUT COMMOND --ADDR RESET
        LDA #$01      ;LOAD COMMEND ADDR_RESET
        STA PORTA
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
;RELOAD VALUE=30000 OR #7530 IN HEX

    ;OUTPUT DATA1
        LDA #$30      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
        BSET 1,PORTF  ;C/D HIGH
    ;OUTPUT DATA2
        LDA #$75      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF  ;WRITE HIGH
        BSET 1,PORTF  ;C/D HIGH
    ;OUTPUT DATA3
        LDA #$00      ;LOAD INITIAL DATA
        STA PORTA
        BCLR 1,PORTF  ;C/D LOW
        BCLR 3,PORTF  ;WRITE LOW
```

```
          STA PORTA     ;RELOAD PORTA FOR DELAY
          BSET 3,PORTF  ;WRITE HIGH
          BSET 1,PORTF  ;C/D HIGH
        ;OUTPUT COMMOND --PRESET RESET
          LDA #$08      ;LOAD COMMEND PRESET_RESET
          STA PORTA
          BCLR 3,PORTF  ;WRITE LOW
          STA PORTA     ;RELOAD PORTA FOR DELAY
          BSET 3,PORTF  ;WRITE HIGH
         ;OUTPUT COMMOND --LATCH COUNT
          LDA #$02      ;LOAD COMMEND LATCH COUNT
          STA PORTA
          BCLR 3,PORTF  ;WRITE LOW
          STA PORTA     ;RELOAD PORTA FOR DELAY
          BSET 3,PORTF  ;WRITE HIGH
        ;OUTPUT COMMOND --ADDR RESET
          LDA #$01      ;LOAD COMMEND ADDR_RESET
          STA PORTA
          BCLR 3,PORTF  ;WRITE LOW
          STA PORTA     ;RELOAD PORTA FOR DELAY
          BSET 3,PORTF  ;WRITE HIGH
; ************** ENCODER1 OFF*************
          LDA #$FF
          STA DDRF      ;SET PORTF OUTPUT
          LDA #$3F      ;SELECT CS2 HIGH,OTHER HIGH
          STA PORTF     ;ENCODER CS2 OFF
          lda #$01
          sta INI_FLAG  ;set initial flag 0
          bset 1,ISCR   ;disable external interrupt

;************ ENCODER 2 INITIALIZATION END*********

          MOV #$7F,PWMOUT ;CONFIG PWM AS GENERAL PURPOSE OUTPUT, SET ALL PINS
ARE HIGH
          CLI
ENTRY:
MAIN:
          CLR R_W
          CLR DEV_ADDR
          CLR WR_BYTE1;
          CLR WR_BYTE2;
          CLR WR_BYTE3;
          CLR NUM_BIT
          CLR REG1;
          CLRA
          CLRX

MAIN_LOOP:
          CLRA
          STA COPCTL
          BCLR 5,DDRC   ;SET PORTC, 5 AS THE INPUT=SCL
          BCLR 6,DDRC   ;SET PORTC,6 AS THE INPUT=SDA
          JSR WRITE_DAC
          lda INI_FLAG
          CMP #$00
          beq COUNT
```

```
        bset 2,ISCR
        bclr 1,ISCR
        bclr 2,ISCR
        lda #$00
        STA INI_FLAG

COUNT:

        JMP MAIN_LOOP
;*************************************************************************
INT:


GET_ADR:
;*****************************************************************************
;         NOW BEGIN RECEIVING THE SLAVE ADDRESS
;*****************************************************************************

        CLRA
        BCLR 5,DDRC    ;SET PORTC, 5 AS THE INPUT=SCL
        BCLR 6,DDRC    ;SET PORTC,6 AS THE INPUT=SDA

        MOV ADSCR,REG1       ;READ ADSCR REG
        BCLR 7,REG1          ;CLEAR COCO BIT IN ADSCR
BIT1_7:
        BRSET 5,PORTC,BIT1_7   ;WAIT FOR SCL FALLING EDGE
        MOV REG1,ADSCR       ;WRITE ADSCR TO START AD CONVERSION?
BIT1_7A:    BRCLR 5,PORTC,BIT1_7A  ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_7   ;READ BYTE1/BIT7
        ADD #$01             ;READ SDA=1
JMP1_7:
        LSLA              ;READ SDA=0
        ;*******BYTE1/BIT7 WAS RECEIVED
W1_7:
        BRSET 5,PORTC,W1_7
BIT1_6:
        BRCLR 5,PORTC,BIT1_6   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_6   ;READ BYTE1/BIT7
        ADD #$01
JMP1_6:
        LSLA              ;READ SDA=0
        ;*******BYTE1/BIT6 WAS RECEIVED

W1_6:
        BRSET 5,PORTC,W1_6
BIT1_5:
        BRCLR 5,PORTC,BIT1_5   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_5   ;READ BYTE1/BIT7
        ADD #$01
JMP1_5:
        LSLA              ;READ SDA=0
        ;*******BYTE1/BIT5 WAS RECEIVED
W1_5:
        BRSET 5,PORTC,W1_5
BIT1_4:
        BRCLR 5,PORTC,BIT1_4   ;WAIT FOR SCL RISING EDGE
```

```
        BRCLR 6,PORTC,JMP1_4   ;READ BYTE1/BIT7
        ADD #$01
JMP1_4:
        LSLA            ;READ SDA=0
        ;*******BYTE1/BIT4 WAS RECEIVED
W1_4:
        BRSET 5,PORTC,W1_4
BIT1_3:
        BRCLR 5,PORTC,BIT1_3   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_3   ;READ BYTE1/BIT7
        ADD #$01
JMP1_3:
        LSLA            ;READ SDA=0
        ;*******BYTE1/BIT3 WAS RECEIVED
W1_3:
        BRSET 5,PORTC,W1_3
BIT1_2:
        BRCLR 5,PORTC,BIT1_2   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_2   ;READ BYTE1/BIT7
        ADD #$01
JMP1_2:
        LSLA            ;READ SDA=0
        ;*******BYTE1/BIT2 WAS RECEIVED
W1_2:
        BRSET 5,PORTC,W1_2
BIT1_1:
        BRCLR 5,PORTC,BIT1_1   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_1   ;READ BYTE1/BIT7
        ADD #$01
JMP1_1:
        LSLA            ;READ SDA=0
        ;*******BYTE1/BIT1 WAS RECEIVED
W1_1:
        BRSET 5,PORTC,W1_1
        STA DEV_ADDR
        ;MOV ADR,RD_BYTE
        BCLR 0,R_W
BIT1_0:
        BRCLR 5,PORTC,BIT1_0   ;WAIT FOR SCL RISING EDGE
        BRCLR 6,PORTC,JMP1_0   ;READ BYTE1/BIT7
        BSET 0,R_W
JMP1_0:
                    ;READ SDA=0
        ;*******BYTE1/BIT0 WAS RECEIVED
        CMP #$42
        BEQ SLAVE_ADR_ACK
       JMP ADR_MISS
;*************************************************************
;       TRANSMIT DATA (MASTER READ)
;*************************************************************
IC_MASTER_READ_START:
          CLRX          ;CLEAR THE DATA POINT X=0
IC_MASTER_READ:
        LDA READ_BYTE,X
        STA RD_BYTE
        BSET 6,DDRC
```

```
MS_RD:
        BRSET 5,PORTC,MS_RD     ;WAIT FOR SCL=0
        BRCLR 7,RD_BYTE,TX07    ;RD_BYTE IS THE BYTE TO SEND TO MASTER
        BSET 6,PORTC
TX07:       BRCLR 5,PORTC,TX07      ;WAIT FOR SCL=1
        STA COPCTL
TX07A:
        BRSET 5,PORTC,TX07A     ;WAIR FOR SCL=0
        ;**************WRITE BIT 7
        BCLR 6,PORTC            ;CLEAR SDA
        BRCLR 6,RD_BYTE,TX06
        BSET 6,PORTC
TX06:       BRCLR 5,PORTC,TX06      ;WAIT FOR SCL=1
        STA COPCTL
TX06A:
        BRSET 5,PORTC,TX06A     ;WAIT FOR SCL=0
        ;**************WRITE BIT 6
        BCLR 6,PORTC            ;CLEAR SDA
        BRCLR 5,RD_BYTE,TX05    ; changed fron 3 to 5
        BSET 6,PORTC
TX05:       BRCLR 5,PORTC,TX05      ;WAIT FOR SCL=1
        STA COPCTL
TX05A:
        BRSET 5,PORTC,TX05A     ;WAIT FOR SCL=0
        ;**************WRITE BIT 5
        BCLR 6,PORTC            ;CLEAR SDA
        BRCLR 4,RD_BYTE,TX04
        BSET 6,PORTC
TX04:       BRCLR 5,PORTC,TX04      ;WAIT FOR SCL=1
        STA COPCTL
TX04A:
        BRSET 5,PORTC,TX04A     ;WAIT FOR SCL=0
        ;**************WRITE BIT 4
        BCLR 6,PORTC            ;CLEAR SDA
        BRCLR 3,RD_BYTE,TX03
        BSET 6,PORTC
TX03:       BRCLR 5,PORTC,TX03      ;WAIT FOR SCL=1
        ;**************WRITE BIT 1
        BCLR 6,PORTC            ;CLEAR SDA
        BRCLR 0,RD_BYTE,TX00
        BSET 6,PORTC
TX00:       BRCLR 5,PORTC,TX00      ;WAIT FOR SCL=1
        STA COPCTL
TX00A:
        BRSET 5,PORTC,TX00A     ;WAIT FOR SCL=0
;**************WRITE BIT0
        BCLR 6,PORTC
;******************READ AXKNOWLEDGE FROM MASTER
        BCLR 5,DDRC
        BCLR 6,DDRC
        STA COPCTL
MST_ACK:
         BRCLR 5,PORTC,MST_ACK  ;WAIT FOR SCL=1
                        ;READ MASTER ACKNOWLEDGE
         BRSET 6,PORTC,MST_ACK1  ;IF SDA=1 NO ACK
         INCX               ;INCREASE POINT TO NEXT BYTE
```

```
        JMP IC_MASTER_READ     ;IF ACK TRANSFER (NEXT) DATA
MST_ACK1:

MST_ACK2:
        BRSET 5,PORTC,MST_ACK2  ;WAIT FOR SCL=0
        ;STA COPCTL
        JMP WAIT_STOP
;*************************************************************************
;       IC_MASTER WRITE
;*************************************************************************
IC_MASTER_WRITE_START:
        CLRX        ;CLEAN BYTE POINTER X=0
IC_MASTER_WRITE:

MW_SL:
        BRSET 5,PORTC,MW_SL    ;WAIT FOR SCL=0
        BCLR 6,DDRC          ;REMOVE ACKNOWLEDGE FROM SDA
MW_SH:
        BRCLR 5,PORTC,MW_SH    ;WAIT FOR SCL=1

        LDA PORTC            ;NEW SAMPLE
        AND #$60            ;MASK OUT SDA AND SCL
        STA TEMP             ;IF NO CHANGE
DO_MW:
        LDA PORTC
        AND #$60
        STA ETEMP
        CMP TEMP
        BEQ DO_MW            ;WAIT NO CHANGE

        BRSET 2,ETEMP,MW_SKP   ;IF SCL CHANGE TO LOW
        JMP RECEIVE_DATA       ;GO TO SKIP BYTE
MW_SKP:
        BRSET 3,ETEMP,MW_SKP2
        JMP GET_ADR
MW_SKP2:
        JMP TWI_STOP
RECEIVE_DATA:
        CLRA
        BRCLR 3,TEMP,BIT2_7
        ADD #$01
BIT2_7:
        LSLA
        ;********BYTE1/BIT7 WAS RECEIVED
W2_7:
        BRSET 5,PORTC,W2_7     ;WAIT FOR SCL=0
BIT2_6:
        BRCLR 5,PORTC,BIT2_6   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_6   ;READ BYTE1/BIT7
        ADD #$01             ;READ SDA=1
JMP2_6:
        LSLA
        ;********BYTE1/BIT6 WAS RECEIVED
W2_6:
        BRSET 5,PORTC,W2_6     ;WAIT FOR SCL=0
BIT2_5:
```

```
        BRCLR 5,PORTC,BIT2_5   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_5   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_5:
        LSLA
        ;********BYTE1/BIT5 WAS RECEIVED
W2_5:
        BRSET 5,PORTC,W2_5     ;WAIT FOR SCL=0
BIT2_4:
        BRCLR 5,PORTC,BIT2_4   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_4   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_4:
        LSLA
        ;********BYTE1/BIT4 WAS RECEIVED
W2_4:
        BRSET 5,PORTC,W2_4     ;WAIT FOR SCL=0
BIT2_3:
        BRCLR 5,PORTC,BIT2_3   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_3   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_3:
        LSLA
        ;********BYTE1/BIT3 WAS RECEIVED
W2_3:
        BRSET 5,PORTC,W2_3     ;WAIT FOR SCL=0
BIT2_2:
        BRCLR 5,PORTC,BIT2_2   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_2   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_2:
        LSLA
        ;********BYTE1/BIT2 WAS RECEIVED
W2_2:
        BRSET 5,PORTC,W2_2     ;WAIT FOR SCL=0
BIT2_1:
        BRCLR 5,PORTC,BIT2_1   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_1   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_1:
        LSLA
        ;********BYTE1/BIT1 WAS RECEIVED
W2_1:
        BRSET 5,PORTC,W2_1     ;WAIT FOR SCL=0
BIT2_0:
        BRCLR 5,PORTC,BIT2_0   ;WAIT FOR SCL=1
        BRCLR 6,PORTC,JMP2_0   ;READ BYTE1/BIT7
        ADD #$01               ;READ SDA=1
JMP2_0:
        LSLA
;******************BYTE1/BIT0 WAS RECEIVED
SLAVE_REC_ACK1:
SL_ACK_RA1:
        BRSET 5,PORTC,SL_ACK_RA1      ;WAIT FOR SCL=0
SLACK1:
        BCLR 6,PORTC
```

```
        BSET 6,DDRC
SL_ACK_RB1:
        BRCLR 5,PORTC,SL_ACK_RB1      ;WAIT FOR SCL=1
        ;BCLR 3,DDRA                  ;SET TO BE INPUT OF SDA
;**********************END OF ACK************************
;               STORE DATA
;*************************************************************
        STA WRITE_BYTE,X              ;SVAE THE RECEIVED DATA

        INCX                          ;INCREAE POINTER
        JMP IC_MASTER_WRITE           ;CONTINUE TO NEXT BYTE
ADR_MISS:
;****************** DROP ACKNOWLEDGE*************************
;     INSTEAL OF HOLD SDA LOW TO ACK MASTER IN 9TH CLOCK CYCLE
;     JUST KEEP SDA CONTROL (HIGH BECAUSE OF PULL UP RESISTER)
;*************************************************************
WHI_DAC:
        BRCLR 5,PORTC,WHI_DAC  ;WAIT FOR SCL HIGH
WLO_DAC:
        BRSET 5,PORTC,WLO_DAC  ;WAIT FOR SCL LOW
;TWI_WAIT_COND:
;WHI_WC:
        ;BRCLR 5,PORTC_WHI_WC  ;WAIT FOR SCL HIGH
        ;BCLR 3,DDRA           ;SET PORTC 6 AS INPUT
        ;BCLR 2,DDRA           ;SET PORTC 5 AS INPUT
        ;LDA PORTA             ;LOAD PORTS
        ;AND #$0C              ;MASK OUT SDA AND SCL
        ;STA TEMP
;DO_WC:
        ;LDA PORTA             ;NEW SAMPLE
        ;AND #$0C              ;MASK OUT SDA AND SCL
        ;CMP TEMP              ;IF NO CHANGE
        ;BEQ DO_WC
        ;STA ETEMP
        ;BRSET 2,ETEMP,SKIP1   ;IF SCL CHANGE TO LOW
        ;JMP TWI_SKIP_BYTE     ;GOTO SKIP BYTE
;SKIP1:
        ;BRSET 3,ETEMP,SKP2    ;IF SDA CHANGE TO LOW
        ;JMP GET_ADR           ;GOTO REPEAT START
;SKP2:
WAIT_STOP:
        STA COPCTL
        BCLR 6,DDRC    ;SET PORTC 6 AS INPUT
        BCLR 5,DDRC    ;SET PORTC 5,AS INPUT
        LDA PORTC      ;LOAD PORTC
        AND #$60       ;MASK OUT SDA AND SCL
        CMP #$20       ;WAIT SCL HIGH
        BNE WAIT_STOP
WAIT_STO:
        STA COPCTL
        LDA PORTC
        AND #$60       ;MASK SCL AND SDA
        CMP #$60       ;IF SCL=1 AND SDA=1
        BNE WAIT_STO
TWI_STOP:
       ;bset 7,TBSC0
```

```
        ;LDA TBCNTL
        ;STA TBCH0L
        ;LDA TBCNTH
        ;STA TBCH0H
        ;lda #$70    ;intrval 0.1ms counter value=368
        ;ADD TBCH0L  ; BEEPER = TACH0L * 1.085 uSECS = 136u
        ;STA TEMPO1 ; THEREFORE BEEPER FREQ = 1/[2*125*1.085uSEC]=3.8 KHZ
        ;LDA #01H
        ;ADC TBCH0H
        ;STA TBCH0H
        ;LDA TEMPO1
        ;STA TBCH0L
        ;BCLR 7,TBSC0
        bset 2,ISCR

        bclr 1,ISCR
        bclr 2,ISCR
         RTI

GET_ENCODER:


        ;ADD TBCH0L  ; BEEPER = TACH0L * 1.085 uSECS = 136u
        ;STA TEMPO1 ; THEREFORE BEEPER FREQ = 1/[2*125*1.085uSEC]=3.8 KHZ
        ;LDA #01H


        ; BCLR 7,TBSC0
;***************TEST CODE********************
        ;LDA #0FFH
        ;STA DDRB
        ;LDA PORTB
        ;EOR #$01
        ;STA PORTB
;****************END TEST CODE*************
;***************SELECT ENCODER 1 ON***************
         LDA #$FF
         STA DDRF     ;SET PORTF OUTPUT
         LDA #$1F     ;SELECT CS1 LOW,OTHER HIGH
         STA PORTF    ;ENCODER CS1 ON
         LDA #$FF
         STA DDRA     ;SET PORTA OUTPUT
        ;OUTPUT COMMOND --LATCH COUNT
         LDA #$02     ;LOAD COMMEND LATCH COUNT
         STA PORTA
         BCLR 3,PORTF  ;WRITE LOW
         STA PORTA    ;RELOAD PORTA FOR DELAY
         BSET 3,PORTF  ;WRITE HIGH
       ;OUTPUT COMMOND --ADDR RESET
         LDA #$01     ;LOAD COMMEND ADDR_RESET
         STA PORTA
         BCLR 3,PORTF  ;WRITE LOW
         STA PORTA    ;RELOAD PORTA FOR DELAY
         BSET 3,PORTF  ;WRITE HIGH
         LDA #$00
         STA DDRA      ;SET PORTA INPUT
```

```
        CLRX
      ;INPUT DATA1
        BCLR 1,PORTF
        BCLR 2,PORTF
        LDA PORTA
        STA READ_BYTE,X
        BSET 2,PORTF
        BSET 1,PORTF
        INCX
      ;INPUT DATA2
      ;INPUT DATA3
        BCLR 5,PORTF
        BCLR 4,PORTF
        LDA PORTA
        STA READ_BYTE,X

; ************** ENCODER1 OFF*************
        LDA #$FF
        STA DDRF      ;SET PORTF OUTPUT
        LDA #$3F      ;SELECT CS1 HIGH,OTHER HIGH
        STA PORTF     ;ENCODER CS1 OFF
;*****************END OF ENCODER 1*********************
;***************SELECT ENCODER 2 ON***************
        LDA #$FF
        STA DDRF      ;SET PORTF OUTPUT
        LDA #$2F      ;SELECT CS1 LOW,OTHER HIGH
        STA PORTF     ;ENCODER CS1 ON
        LDA #$FF
        STA DDRA      ;SET PORTA OUTPUT
       ;OUTPUT COMMOND --LATCH COUNT
        LDA #$02      ;LOAD COMMEND LATCH COUNT
        STA PORTA
        BCLR 3,PORTF   ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF   ;WRITE HIGH
      ;OUTPUT COMMOND --ADDR RESET
        LDA #$01      ;LOAD COMMEND ADDR_RESET
        STA PORTA
        BCLR 3,PORTF   ;WRITE LOW
        STA PORTA     ;RELOAD PORTA FOR DELAY
        BSET 3,PORTF   ;WRITE HIGH
        LDA #$00
        STA DDRA       ;SET PORTA INPUT
        INCX
      ;INPUT DATA1
        BCLR 1,PORT
        INCX
      ;INPUT DATA2
        BCLR 1,PORTF
        BCLR 2,PORTF
        LDA PORTA
        STA READ_BYTE,X
        BSET 2,PORTF
        BSET 1,PORTF
        INCX
      ;INPUT DATA3
```

```
        BCLR 1,PORTF
        BCLR 2,PORTF
        LDA PORTA
        STA READ_BYTE,X
        BSET 5,PORTF
        BSET 2,PORTF
; ************** ENCODER1 OFF*************
        LDA #$FF
        STA DDRF      ;SET PORTF OUTPUT
        LDA #$3F      ;SELECT CS1 HIGH,OTHER HIGH
        STA PORTF     ;ENCODER CS1 OFF
;*****************END OF ENCODER 2*********************

        RTS


;*****************************************************
;       DAC CONVERTER
;       PWM3---WR
;       PWM4---CSLSB
;       PWM5---LDAC
;       PWM6---CSMSB
;
;*****************************************************
WRITE_DAC:
        LDA #$FF
        STA DDRE      ;SET PORTE IS OUTPUT
        CLRA
        STA DDRB
        BCLR 0,DDRC
        BCLR 5,DDRC
        BCLR 6,DDRC
        BCLR 5,DDRC
        LDA PORTC
        AND #$0F
        STA DAC_H
        LDA PORTB
        STA DAC_L
        BSET 3,PWMOUT   ;LOW THE CSLSB PIN
        BSET 2,PWMOUT   ;LOW WRITE PIN
        MOV DAC_L,PORTE
        BCLR 2,PWMOUT   ;HIGH WRITE PIN
        BCLR 3,PWMOUT   ;HIGH CSLSB PIN


        BSET 5,PWMOUT   ;LOW THE CSMSB PIN
        BSET 4,PWMOUT   ;LOW THE LDAC PIN
        BSET 2,PWMOUT   ;LOW WRITE PIN
        MOV DAC_H,PORTE
        BCLR 2,PWMOUT   ;HIGH WRITE PIN
        BCLR 4,PWMOUT   ;LOW THE LDAC PIN
        BCLR 5,PWMOUT   ;HIGH CSMSB PIN
        RTS
```

```
TIMB_CH0:
        BCLR 7,TBSC0
        rti


SCI_TXM:
SCI_REC:
SCI_ERR:
SPI_TXM:
SPI_REC:
AD_CNVT:
TIMB_OVFLOW:
TIMB_CH1:
;TIMB_CH0:
TIMA_OVFLOW:
TIMA_CH3:
TIMA_CH2:
TIMA_CH1:
TIMA_CH0:
PWM_INT:
FAULT4:
FAULT3:
FAULT2:
FAULT1:
PLL_INT:
SWI_INT:
;INT:
          RTI

   ORG $FFD2
   DW SCI_TXM
   DW SCI_REC
   DW SCI_ERR
   DW SPI_TXM
   DW SPI_REC
   DW AD_CNVT
   DW TIMB_OVFLOW
   DW TIMB_CH1
   DW TIMB_CH0        ;IN USE
   DW TIMA_OVFLOW
   DW TIMA_CH3
   DW TIMA_CH2
   DW TIMA_CH1
   DW TIMA_CH0
   DW PWM_INT
   DW FAULT4
   DW FAULT3
   DW FAULT2
   DW FAULT1
   DW PLL_INT
   DW INT          ;IN USE
   DW SWI_INT
   DW RESET        ;IN USE
```

Appendix B

```
/*************************************************************
        Pendubot control algorithm for balancing at top position
                                li Zhang
                                version 0.1
                                March 15,2004
 change history
        1) Kp=58,Kd=9.3;            //can not balance at top
        2) Kp=59.Kd=9.3;            //can balance at top
        2) Kp=60.Kd=9.3;            //can balance at top

*************************************************************/
# include<windows.h>
# include<stdio.h>
# include "UsbI2cIo.h"
#include <iostream.h>
#include "conio.h"
#include <math.h>

#define PI 3.1415926
#define HALFPI 1.5707963
#define ENCDIV 5000         //count per revolution,X4 setting used on encoder
#define ENCINIT 30000L
#define G 9.804         //acceleration of gravity inches/s/s

I2C_TRANS TransI2C;
void _cdecl main(argc,argv)
        int argc;
        char *argv[];

{

        int firstime;
        int cat; //catch
        int pd;
        int al;
        unsigned long istat;
        int digout=2048;
        float diff=0.0;
        float f_digout=0.0;
        float u=0.0;
        float t=0.0;
        long lCntA,lCntB;
        short mcr_cmd=0,icr_cmd=0,occr_cmd=0;
```

```
short sIRQNum;
FILE *fptr;
float link1[1000];
float link2[1000];
float vlink1[1000];
float vlink2[1000];
float tau[1000];
float tsec[1000];
float K1=-16.4615,K2=-3.1287;       //balancing control gains
float K3=-16.2422,K4=-2.0658;       //u=-Kx
float Kp=60.0;                              //swing up outer control gains
float Kd=9.3;
float x1k=0.0;
float x3k=0.0;
float x1old=0.0;
float x3old=0.0;
float x2k=0.0,x4k=0.0;
float x2old[2]={0.0,0.0};
float x4old[2]={0.0,0.0};
float P[5]={0.0308,0.0106,0.0095,0.2087,0.0630}; //identified parameters
float d11,d12or21,d22;                      //paritial feedback linearization gains
float c11,c12,c21;
float phi[2];
float v1,dbar,c1bar,c2bar,gbar,h;
float dither_w=20.0,dither_ampl=0.25; //dither input to get ride of some

float sample;


int i=0;
int j=0;
int t_h,t_l,t_h_old,t_l_old;
unsigned long voltage;
int n;
        int time,time_new,time_old;

HANDLE hDevInstance=INVALID_HANDLE_VALUE;
hDevInstance=DAPI_OpenDeviceInstance("UsbI2cIo",0);
if(argc==3)
{
        Kp=(float) atof(argv[1]);
        Kd=(float) atof(argv[2]);
}
printf("Kp=%f,Kd=%f\n",Kp,Kd);
printf("Please hold main switch and press any key\n");
do{
```

```
}while(!kbhit()); //enddo

if(!getch())
        (void)getch();
printf("\n");
printf("press anyKey to Stop Control\n");

x1old=-HALFPI;      //initialize position
x3old=0.0;
x2old[0]=0.0;
x4old[0]=0.0;
x2old[1]=0.0;
x4old[1]=0.0;
firstime=1;
pd=1;
cat=0;
i=0;
sample=0.0;
t_h=0;
t_l=0;



if(hDevInstance!=INVALID_HANDLE_VALUE)
{


        printf("Opened device UsbI2cIo0\n");

//LONG lWriteCnt;
DAPI_ConfigIoPorts(hDevInstance,0x00000000);
// initialize I2C transaction structure
TransI2C.byTransType = I2C_TRANS_NOADR;
TransI2C.wMemoryAddr = 0;
TransI2C.bySlvDevAddr = 0x42;  // I2C device Id (R/W bit is handled by API
function)
TransI2C.wCount = 8;     // number of bytes to write
TransI2C.Data[0] = 0x11;  // Data byte 0 value
TransI2C.Data[1] = 0x22;  // Data byte 1 value
TransI2C.Data[2] = 0x33;  // Data byte 2 value
TransI2C.Data[3] = 0x44;  // Data byte 0 value
TransI2C.Data[4] = 0x55;  // Data byte 1 value
TransI2C.Data[5] = 0x66;
TransI2C.Data[6] = 0x77;  // Data byte 1 value
TransI2C.Data[7] = 0x88;
// call the API funciton to perform the I2C write transaction
```

```
//

digout=2048;                    //set the output voltage of the DAC to initially
DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF); //output voltage=0


        while(!kbhit())
        {
        //config all I/O piorts for output

//      printf("Configured all porrts asd outputs\n");
        //DAPI_ConfigIoPorts(hDevInstance,0x00000000);
        //clear all bits except A.0 which will be set
         if(firstime)    //first time in the control loop start the sampling
         {
                firstime=0;
         }

DAPI_ReadI2c(hDevInstance, &TransI2C); //read encoder count and time sample


                t_h_old=t_h;            //calculate the sample time interval
                t_l_old=t_l;
                time_old=t_h_old+t_l_old;
//printf("Cleared all ports bits to 0,with the exception of A0 which was set to 1\n");

        //now st bit B7 and clear bit A0 leave others bits unmodified
   //DAPI_WriteIoPorts(hDevInstance,voltage,0xFFFFFFFF);
           t_h=TransI2C.Data[6]*255;
                t_l=TransI2C.Data[7];
                time_new=t_h+t_l;

                lCntA=TransI2C.Data[0]+TransI2C.Data[1]*256;
                lCntB=TransI2C.Data[3]+TransI2C.Data[4]*256;
                x1k=(lCntA-ENCINIT)*((2*PI)/ENCDIV)-HALFPI;
//convert count to radians
                x3k=(lCntB-ENCINIT)*((2*PI)/ENCDIV);
                x2k=(x1k-x1old)/sample;      //calculate velocity
                x4k=(x3k-x3old)/sample;      //calculate velocity
                x2k=(x2k+x2old[0]+x2old[1])/3.0; //average last 3 velocities to get
                                                 //ride of numerical noise
                x4k=(x4k+x4old[0]+x4old[1])/3.0;
```

```
printf("digout=%f\n",x1k);
if(!cat) //if links have not come in range to balance
{
        if(fabs(x1k-HALFPI)<0.10) //if theta1=0
        {
                if(fabs(x3k)<0.20)      //if theta2 less than 0.2
                {
                u=-K1*(x1k-HALFPI)-K2*x2k-K3*x3k-K4*x4k;
                //calculate contorl
                        if(fabs(u)<6)
                        {
                                pd=0;
                                cat=1;
                        }
                }
        }
}


if(pd)          //if sing up control is still in use
{
        //if(t<0.301)
        //{
        //      u=-1.0;
        //}
        //else
        //{
                d11=P[0]+P[1]+2*P[2]*cos(x3k);
                d12or21=P[1]+P[2]*cos(x3k);
                d22=P[1];       //calulate lagrangain dynamics
                h=-P[2]*sin(x3k);
                c11=h*x4k;
                c12=h*x4k+h*x2k;
                c21=-h*x2k;

                phi[0]=P[3]*G*cos(x1k)+P[4]*G*cos(x1k+x3k);
                phi[1]=P[4]*G*cos(x1k+x3k);
                dbar=d11-(d12or21*d12or21/d22);
                c1bar=c11-(d12or21*c21/d22);
                c2bar=c12;
                gbar=phi[0]-(d12or21*phi[1]/d22);
                //outer loop contrl
                v1=-Kd*(x2k)+Kp*(HALFPI-x1k);
                /*inner loop control*/
                u=dbar*v1+c1bar*x2k+c2bar*x4k+gbar;
```

```
        //}
    }
    else    //perform balanceing control
    {
        u=-K1*(x1k-HALFPI)-K2*x2k-K3*x3k-K4*x4k;
        if(fabs(x1k-HALFPI)<0.2)
        {

        u=u+dither_ampl*sin(dither_w*t);  //add dither signal
        }
        if(fabs(u)>6) //if control gets too large
        {                       //switch back to swing up control
                                // which is servoing around a seyt
point
            u=0.0; //set control to sero;
            pd=1;
        }
    }

    if(u>9.95)
        u=9.95;
    if(u<-9.95)
        u=-9.95;
    printf("u=%f\n",u);

    f_digout=2048+204.8*u;

    digout=(int)f_digout;
    diff=f_digout-(float)digout;
    if(diff>0.5)
        digout++;

        //limit output
        if(digout>4085)
        {
            digout=4085;
        }
        if(digout<11)
        {
            digout=11;
        }



        DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF);
    //output D/A voltage
```

```
                x2old[1]=x2old[0];
                x4old[1]=x4old[0];
                x1old=x1k;
                x3old=x3k;
                x2old[0]=x2k;
                x4old[0]=x4k;

                if((i<1000))    //save the first 1000 point
                {
                        link1[i]=x1k;
                        link2[i]=x3k;
                        vlink1[i]=x2k;
                        vlink2[i]=x4k;
                        tau[i]=u;
                        tsec[i]=t;


                        i++;
                }

                t=t+sample;
                printf("x1k= %f %f\n",vlink2[i-1],t);

        for(j=0;j<8;j++)
        {
        printf("data[ %d ]= %d\n",j,TransI2C.Data[j]);
        }

//      printf("voltage=%d\n",voltage);



}

digout=2048;            //set DAC output to zero

DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF); //output D/A voltage
        if((fptr=fopen("data.m","w"))==NULL)
        {
                printf("Error opening file\n");
        }
        fprintf(fptr,"Y=["); //write data to a file in Matlab M-file
        for(i=0;i<1000;i++)
        {
fprintf(fptr,"%f %f %f %f %f%f\n",tsec[i],link1[i],link2[i],vlink1[i],vlink2[i],tau[i]);
        }
```

```
                fprintf(fptr,"];\n");
                fclose(fptr);
                printf("output complate\n");



        }
        else
        {

                printf("failed to open a handle to device UsbI2cIo\n");
        }

        if(hDevInstance!=INVALID_HANDLE_VALUE)
        {       //close handle to device instance
                DAPI_CloseDeviceInstance(hDevInstance);
        }

}
```

Appendix C

```
/*************************************************************
        Pendubot control algorithm for balancing at mid position
                                li Zhang
                                version 0.1
                                March 15,2004
 change history
        1)      Kp=150,Kd=21.0;

*************************************************************/
# include<windows.h>
# include<stdio.h>
# include "UsbI2cIo.h"
#include <iostream.h>
#include "conio.h"
#include <math.h>

#define PI 3.1415926
#define HALFPI 1.5707963
#define ENCDIV 5000          //count per revolution,X4 setting used on encoder
#define ENCINIT 30000L
#define G 9.804         //acceleration of gravity inches/s/s

I2C_TRANS TransI2C;
void _cdecl main(argc,argv)
        int argc;
        char *argv[];

{

        int firstime;
        int cat; //catch
        int pd;
        int al;
        unsigned long istat;
        int digout=2048;
        float diff=0.0;
        float f_digout=0.0;
        float u=0.0;
        float t=0.0;
        long lCntA,lCntB;
        short mcr_cmd=0,icr_cmd=0,occr_cmd=0;
        short sIRQNum;
        FILE *fptr;
```

```
float link1[1000];
float link2[1000];
float vlink1[1000];
float vlink2[1000];
float tau[1000];
float tsec[1000];
float K1=7.246,K2=0.7509;  /* balancing control gains */
float K3=10.1643,K4=1.2591;  /* u=-Kx */
float Kp=150.0;                                    //swing up outer control gains
float Kd=21.0;
float x1k=0.0;
float x3k=0.0;
float x1old=0.0;
float x3old=0.0;
float x2k=0.0,x4k=0.0;
float x2old[2]={0.0,0.0};
float x4old[2]={0.0,0.0};
float P[5]={0.0308,0.0106,0.0095,0.2087,0.0630}; //identified parameters
float d11,d12or21,d22;                          //paritial feedback linearization gains
float c11,c12,c21;
float phi[2];
float v1,dbar,c1bar,c2bar,gbar,h;
float dither_w=4.5,dither_ampl=1.165; //dither input to get ride of some
float q1d[3],ts;
float sample;

float w=4.5,ampl=1.165; /* swing up trajectory u=ampl*sin(w*t) */




int i=0;
int j=0;
int t_h,t_l,t_h_old,t_l_old;
unsigned long voltage;
int n;
        int time,time_new,time_old;

HANDLE hDevInstance=INVALID_HANDLE_VALUE;
hDevInstance=DAPI_OpenDeviceInstance("UsbI2cIo",0);
if(argc==3)
{
        Kp=(float) atof(argv[1]);
        Kd=(float) atof(argv[2]);
}
printf("Kp=%f,Kd=%f\n",Kp,Kd);
```

```
        printf("Please hold main switch and press any key\n");
        do{
        }while(!kbhit()); //enddo

        if(!getch())
                (void)getch();
        printf("\n");
        printf("press anyKey to Stop Control\n");

        x1old=-HALFPI;      //initialize position
        x3old=0.0;
        x2old[0]=0.0;
        x4old[0]=0.0;
        x2old[1]=0.0;
        x4old[1]=0.0;
        firstime=1;
        pd=1;
        cat=0;
        i=0;
        sample=0.0;
        t_h=0;
        t_l=0;
        ts = 2*PI/w;


        if(hDevInstance!=INVALID_HANDLE_VALUE)
        {


                printf("Opened device UsbI2cIo0\n");

        //LONG lWriteCnt;
        DAPI_ConfigIoPorts(hDevInstance,0x00000000);
// initialize I2C transaction structure
TransI2C.byTransType = I2C_TRANS_NOADR;
TransI2C.wMemoryAddr = 0;
TransI2C.bySlvDevAddr = 0x42;  // I2C device Id (R/W bit is handled by API function)
TransI2C.wCount = 8;      // number of bytes to write
TransI2C.Data[0] = 0x11;  // Data byte 0 value
TransI2C.Data[1] = 0x22;  // Data byte 1 value
TransI2C.Data[2] = 0x33;  // Data byte 2 value
TransI2C.Data[3] = 0x44;  // Data byte 0 value
TransI2C.Data[4] = 0x55;  // Data byte 1 value
TransI2C.Data[5] = 0x66;
TransI2C.Data[6] = 0x77;  // Data byte 1 value
TransI2C.Data[7] = 0x88;
```

```
// call the API funciton to perform the I2C write transaction
//

        digout=2048;                    //set the output voltage of the DAC to initially
        DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF); //output voltage=0


                while(!kbhit())
                {
                //config all I/O piorts for output

        //      printf("Configured all porrts asd outputs\n");
                //DAPI_ConfigIoPorts(hDevInstance,0x00000000);
                //clear all bits except A.0 which will be set
                 if(firstime)    //first time in the control loop start the sampling
                 {
                        firstime=0;
                 }

         DAPI_ReadI2c(hDevInstance, &TransI2C);//read encoder count and time sample


                        t_h_old=t_h;            //calculate the sample time interval
                        t_l_old=t_l;
                        time_old=t_h_old+t_l_old;
        //printf("Cleared all ports bits to 0,with the exception of A0 which was set to 1\n");

                //now st bit B7 and clear bit A0 leave others bits unmodified
          //DAPI_WriteIoPorts(hDevInstance,voltage,0xFFFFFFFF);
                  t_h=TransI2C.Data[6]*255;
                        t_l=TransI2C.Data[7];
                        time_new=t_h+t_l;


                        lCntA=TransI2C.Data[0]+TransI2C.Data[1]*256;
                        lCntB=TransI2C.Data[3]+TransI2C.Data[4]*256;
                        x1k=(lCntA-ENCINIT)*((2*PI)/ENCDIV)-HALFPI;
        //convert count to radians
                        x3k=(lCntB-ENCINIT)*((2*PI)/ENCDIV);
                        x2k=(x1k-x1old)/sample;       //calculate velocity
                        x4k=(x3k-x3old)/sample;       //calculate velocity
                        x2k=(x2k+x2old[0]+x2old[1])/3.0; //average last 3 velocities to get
                                                         //ride of numerical noise
                        x4k=(x4k+x4old[0]+x4old[1])/3.0;

                        printf("digout=%f\n",x1k);
```

```
if(!cat) //if links have not come in range to balance
{
        if(fabs(x1k+HALFPI)<0.20) //if theta1=0
        {
                if(fabs(x3k-PI)<0.30) //if theta2 less than 0.2
                {
        u=-K1*(x1k+HALFPI)-K2*x2k-K3*(x3k-PI)-K4*x4k;
                                        //calculate contorl
                        if(fabs(u)<8.0)
                        {
                                pd=0;
                                cat=1;
                        }
                }
        }
}


if(pd)          //if sing up control is still in use
{
        //if(t<0.301)
        //{
        //      u=-1.0;
        //}
        //else
        //{
                d11=P[0]+P[1]+2*P[2]*cos(x3k);
                d12or21=P[1]+P[2]*cos(x3k);
                d22=P[1];       //calulate lagrangain dynamics
                h=-P[2]*sin(x3k);
                c11=h*x4k;
                c12=h*x4k+h*x2k;
                c21=-h*x2k;

                phi[0]=P[3]*G*cos(x1k)+P[4]*G*cos(x1k+x3k);
                phi[1]=P[4]*G*cos(x1k+x3k);
                dbar=d11-(d12or21*d12or21/d22);
                c1bar=c11-(d12or21*c21/d22);
                c2bar=c12;
                gbar=phi[0]-(d12or21*phi[1]/d22);
                if (t<ts) {
                /* Trajectory for Pendubot to follow for swing up */
                        q1d[0] = ampl*sin(w*t)-HALFPI;
                        q1d[1] = w*ampl*cos(w*t);
                        q1d[2] = -w*w*ampl*sin(w*t);
                 } else {
```

```
                        q1d[0] = -HALFPI;
                        q1d[1] = 0.0;
                        q1d[2] = 0.0;
                    } /* endif */


                    //outer loop contrl
            v1 = q1d[2] + Kd*(q1d[1] - x2k) + Kp*(q1d[0]-x1k);
            /*inner loop control*/
            u=dbar*v1+c1bar*x2k+c2bar*x4k+gbar;

            //}
}
else    //perform balanceing control
{
            u=-K1*(x1k+HALFPI)-K2*x2k-K3*(x3k-PI)-K4*x4k;

            if(fabs(u)>8) //if control gets too large
            {       //switch back to swing up control
                    // which is servoing around a seyt point
                    u=0.0;  //set control to sero;
                    pd=1;
            }
}

if(u>9.95)
        u=9.95;
if(u<-9.95)
        u=-9.95;
printf("u=%f\n",u);

f_digout=2048+204.8*u;

digout=(int)f_digout;
diff=f_digout-(float)digout;
if(diff>0.5)
        digout++;

        //limit output
        if(digout>4085)
        {
                digout=4085;
        }
        if(digout<11)
        {
                digout=11;
```

```
            }


        DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF);
            //output D/A voltage
            x2old[1]=x2old[0];
            x4old[1]=x4old[0];
            x1old=x1k;
            x3old=x3k;
            x2old[0]=x2k;
            x4old[0]=x4k;

            if((i<1000))    //save the first 1000 point
            {
                link1[i]=x1k;
                link2[i]=x3k;
                vlink1[i]=x2k;
                vlink2[i]=x4k;
                tau[i]=u;
                tsec[i]=t;


                i++;
            }

            t=t+sample;
            printf("x1k= %f %f\n",vlink2[i-1],t);

        for(j=0;j<8;j++)
        {
        printf("data[ %d ]= %d\n",j,TransI2C.Data[j]);
        }

//      printf("voltage=%d\n",voltage);



}

digout=2048;            //set DAC output to zero

DAPI_WriteIoPorts(hDevInstance,digout,0xFFFFFFFF);
//output D/A voltage
if((fptr=fopen("data.m","w"))==NULL)
{
```

```
                printf("Error opening file\n");
            }
            fprintf(fptr,"Y=["); //write data to a file in Matlab M-file
            for(i=0;i<1000;i++)
            {
fprintf(fptr,"%f %f %f %f %f %f\n",tsec[i],link1[i],link2[i],vlink1[i],vlink2[i],tau[i]);
            }
            fprintf(fptr,"];\n");
            fclose(fptr);
            printf("output complate\n");



    }
    else
    {

            printf("failed to open a handle to device UsbI2cIo\n");
    }

    if(hDevInstance!=INVALID_HANDLE_VALUE)
    {       //close handle to device instance
            DAPI_CloseDeviceInstance(hDevInstance);
    }

}
```